

---

# MCUXpresso SDK API Reference Manual

**NXP Semiconductors**

Document Number: MCUXSDKIMXRT1050APIRM

Rev. 0

May 2018





# Contents

Chapter **Introduction**

Chapter **Driver errors status**

Chapter **Architectural Overview**

Chapter **Trademarks**

Chapter **ADC: 12-bit Analog to Digital Converter Driver**

<b>5.1</b>	<b>Overview</b> . . . . .	<b>13</b>
<b>5.2</b>	<b>Typical use case</b> . . . . .	<b>13</b>
5.2.1	Polling Configuration . . . . .	13
5.2.2	Polling Configuration . . . . .	13
<b>5.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>16</b>
5.3.1	struct adc_config_t . . . . .	16
5.3.2	struct adc_offest_config_t . . . . .	17
5.3.3	struct adc_hardware_compare_config_t . . . . .	17
5.3.4	struct adc_channel_config_t . . . . .	17
<b>5.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>17</b>
5.4.1	FSL_ADC_DRIVER_VERSION . . . . .	17
<b>5.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>18</b>
5.5.1	adc_status_flags_t . . . . .	18
5.5.2	adc_reference_voltage_source_t . . . . .	18
5.5.3	adc_sample_period_mode_t . . . . .	18
5.5.4	adc_clock_source_t . . . . .	18
5.5.5	adc_clock_driver_t . . . . .	19
5.5.6	adc_resolution_t . . . . .	19
5.5.7	adc_hardware_compare_mode_t . . . . .	19
5.5.8	adc_hardware_average_mode_t . . . . .	19
<b>5.6</b>	<b>Function Documentation</b> . . . . .	<b>19</b>
5.6.1	ADC_Init . . . . .	19
5.6.2	ADC_Deinit . . . . .	20

# Contents

Section Number	Title	Page Number
5.6.3	ADC_GetDefaultConfig . . . . .	20
5.6.4	ADC_SetChannelConfig . . . . .	20
5.6.5	ADC_GetChannelConversionValue . . . . .	21
5.6.6	ADC_GetChannelStatusFlags . . . . .	21
5.6.7	ADC_DoAutoCalibration . . . . .	22
5.6.8	ADC_SetOffsetConfig . . . . .	22
5.6.9	ADC_EnableDMA . . . . .	22
5.6.10	ADC_EnableHardwareTrigger . . . . .	23
5.6.11	ADC_SetHardwareCompareConfig . . . . .	23
5.6.12	ADC_SetHardwareAverageConfig . . . . .	23
5.6.13	ADC_GetStatusFlags . . . . .	24
5.6.14	ADC_ClearStatusFlags . . . . .	25
<b>5.7</b>	<b>Variable Documentation . . . . .</b>	<b>26</b>
5.7.1	enableOverWrite . . . . .	26
5.7.2	enableContinuousConversion . . . . .	26
5.7.3	enableHighSpeed . . . . .	26
5.7.4	enableLowPower . . . . .	26
5.7.5	enableLongSample . . . . .	26
5.7.6	enableAsynchronousClockOutput . . . . .	26
5.7.7	referenceVoltageSource . . . . .	26
5.7.8	samplePeriodMode . . . . .	26
5.7.9	clockSource . . . . .	26
5.7.10	clockDriver . . . . .	26
5.7.11	resolution . . . . .	26
5.7.12	enableSigned . . . . .	26
5.7.13	offsetValue . . . . .	26
5.7.14	hardwareCompareMode . . . . .	26
5.7.15	value1 . . . . .	27
5.7.16	value2 . . . . .	27
5.7.17	channelNumber . . . . .	27
5.7.18	enableInterruptOnConversionCompleted . . . . .	27
<b>Chapter</b>	<b>ADC_ETC: ADC External Trigger Control</b>	
<b>6.1</b>	<b>Overview . . . . .</b>	<b>29</b>
<b>6.2</b>	<b>Typical use case . . . . .</b>	<b>29</b>
6.2.1	Software trigger Configuration . . . . .	29
6.2.2	Hardware trigger Configuration . . . . .	29
<b>6.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>30</b>
6.3.1	struct adc_etc_config_t . . . . .	30
6.3.2	struct adc_etc_trigger_chain_config_t . . . . .	30
6.3.3	struct adc_etc_trigger_config_t . . . . .	30

# Contents

Section Number	Title	Page Number
<b>6.4</b>	<b>Macro Definition Documentation</b>	<b>30</b>
6.4.1	FSL_ADC_ETC_DRIVER_VERSION	30
6.4.2	ADC_ETC_DMA_CTRL_TRGn_REQ_MASK	30
<b>6.5</b>	<b>Function Documentation</b>	<b>30</b>
6.5.1	ADC_ETC_Init	30
6.5.2	ADC_ETC_Deinit	30
<b>Chapter</b>	<b>AIPSTZ: AHB to IP Bridge</b>	
<b>7.1</b>	<b>Overview</b>	<b>33</b>
<b>7.2</b>	<b>Enumeration Type Documentation</b>	<b>33</b>
7.2.1	aipstz_master_privilege_level_t	33
7.2.2	aipstz_master_t	34
7.2.3	aipstz_peripheral_access_control_t	34
7.2.4	aipstz_peripheral_t	34
<b>7.3</b>	<b>Function Documentation</b>	<b>34</b>
7.3.1	AIPSTZ_SetMasterPriviledgeLevel	34
7.3.2	AIPSTZ_SetPeripheralAccessControl	34
<b>Chapter</b>	<b>AOI: Crossbar AND/OR/INVERT Driver</b>	
<b>8.1</b>	<b>Overview</b>	<b>37</b>
<b>8.2</b>	<b>Function groups</b>	<b>37</b>
8.2.1	AOI Initialization	37
<b>8.3</b>	<b>Data Structure Documentation</b>	<b>39</b>
8.3.1	struct aoi_event_config_t	39
<b>8.4</b>	<b>Macro Definition Documentation</b>	<b>39</b>
8.4.1	FSL_AOI_DRIVER_VERSION	39
<b>8.5</b>	<b>Enumeration Type Documentation</b>	<b>40</b>
8.5.1	aoi_input_config_t	40
8.5.2	aoi_event_t	40
<b>8.6</b>	<b>Function Documentation</b>	<b>40</b>
8.6.1	AOI_Init	40
8.6.2	AOI_Deinit	40
8.6.3	AOI_GetEventLogicConfig	40
8.6.4	AOI_SetEventLogicConfig	41

# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>BEE: Bus Encryption Engine</b>	
<b>9.1</b>	<b>BEE Driver Initialization and Configuration</b> . . . . .	<b>43</b>
<b>9.2</b>	<b>Enable &amp; Disable BEE</b> . . . . .	<b>43</b>
<b>9.3</b>	<b>Set BEE region config and key</b> . . . . .	<b>43</b>
<b>9.4</b>	<b>Status</b> . . . . .	<b>43</b>
<b>9.5</b>	<b>Get offset</b> . . . . .	<b>44</b>
9.5.1	BEE example . . . . .	44
<b>Chapter</b>	<b>CACHE: CACHE Memory Controller</b>	
<b>10.1</b>	<b>Overview</b> . . . . .	<b>45</b>
<b>10.2</b>	<b>Function groups</b> . . . . .	<b>45</b>
<b>10.3</b>	<b>Macro Definition Documentation</b> . . . . .	<b>46</b>
10.3.1	FSL_CACHE_DRIVER_VERSION . . . . .	46
<b>10.4</b>	<b>Function Documentation</b> . . . . .	<b>47</b>
10.4.1	L1CACHE_InvalidateICacheByRange . . . . .	47
10.4.2	L1CACHE_InvalidateDCacheByRange . . . . .	48
10.4.3	L1CACHE_CleanDCacheByRange . . . . .	48
10.4.4	L1CACHE_CleanInvalidateDCacheByRange . . . . .	49
10.4.5	ICACHE_InvalidateByRange . . . . .	50
10.4.6	DCACHE_InvalidateByRange . . . . .	50
10.4.7	DCACHE_CleanByRange . . . . .	51
10.4.8	DCACHE_CleanInvalidateByRange . . . . .	52
<b>Chapter</b>	<b>CMP: Analog Comparator Driver</b>	
<b>11.1</b>	<b>Overview</b> . . . . .	<b>53</b>
<b>11.2</b>	<b>Typical use case</b> . . . . .	<b>53</b>
11.2.1	Polling Configuration . . . . .	53
11.2.2	Interrupt Configuration . . . . .	53
<b>11.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>55</b>
11.3.1	struct cmp_config_t . . . . .	55
11.3.2	struct cmp_filter_config_t . . . . .	55
11.3.3	struct cmp_dac_config_t . . . . .	56
<b>11.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>56</b>
11.4.1	FSL_CMP_DRIVER_VERSION . . . . .	56

# Contents

Section Number	Title	Page Number
<b>11.5</b>	<b>Enumeration Type Documentation</b>	<b>56</b>
11.5.1	_cmp_interrupt_enable	56
11.5.2	_cmp_status_flags	56
11.5.3	cmp_hysteresis_mode_t	57
11.5.4	cmp_reference_voltage_source_t	57
<b>11.6</b>	<b>Function Documentation</b>	<b>57</b>
11.6.1	CMP_Init	57
11.6.2	CMP_Deinit	57
11.6.3	CMP_Enable	58
11.6.4	CMP_GetDefaultConfig	58
11.6.5	CMP_SetInputChannels	58
11.6.6	CMP_EnableDMA	59
11.6.7	CMP_EnableWindowMode	59
11.6.8	CMP_SetFilterConfig	59
11.6.9	CMP_SetDACConfig	60
11.6.10	CMP_EnableInterrupts	60
11.6.11	CMP_DisableInterrupts	60
11.6.12	CMP_GetStatusFlags	60
11.6.13	CMP_ClearStatusFlags	61
<b>Chapter</b>	<b>CSI: CMOS Sensor Interface</b>	
<b>12.1</b>	<b>Overview</b>	<b>63</b>
<b>12.2</b>	<b>Frame Buffer Queue</b>	<b>63</b>
<b>12.3</b>	<b>Typical use case</b>	<b>63</b>
12.3.1	Receive with functional APIs	63
12.3.2	Receive with transactional APIs	64
<b>12.4</b>	<b>Data Structure Documentation</b>	<b>67</b>
12.4.1	struct csi_config_t	67
12.4.2	struct _csi_handle	68
<b>12.5</b>	<b>Macro Definition Documentation</b>	<b>69</b>
12.5.1	CSI_DRIVER_QUEUE_SIZE	69
<b>12.6</b>	<b>Typedef Documentation</b>	<b>69</b>
12.6.1	csi_transfer_callback_t	69
<b>12.7</b>	<b>Enumeration Type Documentation</b>	<b>69</b>
12.7.1	_csi_status	69
12.7.2	csi_work_mode_t	70
12.7.3	csi_data_bus_t	70
12.7.4	_csi_polarity_flags	70

# Contents

Section Number	Title	Page Number
12.7.5	csi_fifo_t . . . . .	70
12.7.6	_csi_interrupt_enable . . . . .	70
12.7.7	_csi_flags . . . . .	71
<b>12.8</b>	<b>Function Documentation . . . . .</b>	<b>72</b>
12.8.1	CSI_Init . . . . .	72
12.8.2	CSI_Deinit . . . . .	72
12.8.3	CSI_Reset . . . . .	72
12.8.4	CSI_GetDefaultConfig . . . . .	73
12.8.5	CSI_ClearFifo . . . . .	73
12.8.6	CSI_ReflashFifoDma . . . . .	73
12.8.7	CSI_EnableFifoDmaRequest . . . . .	74
12.8.8	CSI_Start . . . . .	74
12.8.9	CSI_Stop . . . . .	74
12.8.10	CSI_SetRxBufferAddr . . . . .	74
12.8.11	CSI_EnableInterrupts . . . . .	75
12.8.12	CSI_DisableInterrupts . . . . .	75
12.8.13	CSI_GetStatusFlags . . . . .	75
12.8.14	CSI_ClearStatusFlags . . . . .	75
12.8.15	CSI_TransferCreateHandle . . . . .	76
12.8.16	CSI_TransferStart . . . . .	76
12.8.17	CSI_TransferStop . . . . .	77
12.8.18	CSI_TransferSubmitEmptyBuffer . . . . .	77
12.8.19	CSI_TransferGetFullBuffer . . . . .	78
12.8.20	CSI_TransferHandleIRQ . . . . .	78
<b>Chapter</b>	<b>DCDC: DCDC Converter</b>	
<b>13.1</b>	<b>Overview . . . . .</b>	<b>79</b>
<b>13.2</b>	<b>Function groups . . . . .</b>	<b>79</b>
13.2.1	Initialization and deinitialization . . . . .	79
13.2.2	Status . . . . .	79
13.2.3	Misc control . . . . .	79
<b>13.3</b>	<b>Application guideline . . . . .</b>	<b>80</b>
13.3.1	Continous conduction mode . . . . .	80
13.3.2	Discontinous conduction mode . . . . .	80
<b>13.4</b>	<b>Data Structure Documentation . . . . .</b>	<b>83</b>
13.4.1	struct dcdc_detection_config_t . . . . .	83
13.4.2	struct dcdc_loop_control_config_t . . . . .	84
13.4.3	struct dcdc_low_power_config_t . . . . .	84
13.4.4	struct dcdc_internal_regulator_config_t . . . . .	84
13.4.5	struct dcdc_min_power_config_t . . . . .	85



# Contents

Section Number	Title	Page Number
<b>13.5</b>	<b>Macro Definition Documentation</b> . . . . .	<b>85</b>
13.5.1	FSL_DCDC_DRIVER_VERSION . . . . .	85
<b>13.6</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>85</b>
13.6.1	_dcdc_status_flags_t . . . . .	85
13.6.2	dcdc_comparator_current_bias_t . . . . .	85
13.6.3	dcdc_over_current_threshold_t . . . . .	85
13.6.4	dcdc_peak_current_threshold_t . . . . .	86
13.6.5	dcdc_count_charging_time_period_t . . . . .	86
13.6.6	dcdc_count_charging_time_threshold_t . . . . .	86
13.6.7	dcdc_clock_source_t . . . . .	86
<b>13.7</b>	<b>Function Documentation</b> . . . . .	<b>86</b>
13.7.1	DCDC_Init . . . . .	86
13.7.2	DCDC_Deinit . . . . .	87
13.7.3	DCDC_GetstatusFlags . . . . .	87
13.7.4	DCDC_EnableOutputRangeComparator . . . . .	87
13.7.5	DCDC_SetClockSource . . . . .	87
13.7.6	DCDC_GetDefaultDetectionConfig . . . . .	88
13.7.7	DCDC_SetDetectionConfig . . . . .	88
13.7.8	DCDC_GetDefaultLowPowerConfig . . . . .	88
13.7.9	DCDC_SetLowPowerConfig . . . . .	89
13.7.10	DCDC_ResetCurrentAlertSignal . . . . .	89
13.7.11	DCDC_SetBandgapVoltageTrimValue . . . . .	89
13.7.12	DCDC_GetDefaultLoopControlConfig . . . . .	89
13.7.13	DCDC_SetLoopControlConfig . . . . .	90
13.7.14	DCDC_SetMinPowerConfig . . . . .	90
13.7.15	DCDC_SetLPComparatorBiasValue . . . . .	90
13.7.16	DCDC_AdjustTargetVoltage . . . . .	91
13.7.17	DCDC_SetInternalRegulatorConfig . . . . .	91
13.7.18	DCDC_EnableAdjustDelay . . . . .	91
13.7.19	DCDC_EnableImproveTransition . . . . .	91
13.7.20	DCDC_BootIntoDCM . . . . .	92
13.7.21	DCDC_BootIntoCCM . . . . .	92
<b>13.8</b>	<b>Variable Documentation</b> . . . . .	<b>93</b>
13.8.1	enableXtalokDetection . . . . .	93
13.8.2	powerDownOverVoltageDetection . . . . .	93
13.8.3	powerDownLowVlotageDetection . . . . .	93
13.8.4	powerDownOverCurrentDetection . . . . .	93
13.8.5	powerDownPeakCurrentDetection . . . . .	93
13.8.6	powerDownZeroCrossDetection . . . . .	93
13.8.7	OverCurrentThreshold . . . . .	93
13.8.8	PeakCurrentThreshold . . . . .	93
13.8.9	enableCommonHysteresis . . . . .	93

# Contents

Section Number	Title	Page Number
13.8.10	enableCommonThresholdDetection . . . . .	93
13.8.11	enableInvertHysteresisSign . . . . .	93
13.8.12	enableRCThresholdDetection . . . . .	93
13.8.13	enableRCScaleCircuit . . . . .	93
13.8.14	complementFeedForwardStep . . . . .	93
13.8.15	controlParameterMagnitude . . . . .	94
13.8.16	integralProportionalRatio . . . . .	94
13.8.17	enableOverloadDetection . . . . .	94
13.8.18	enableAdjustHystereticValue . . . . .	94
13.8.19	countChargingTimePeriod . . . . .	94
13.8.20	countChargingTimeThreshold . . . . .	94
13.8.21	enableLoadResistor . . . . .	94
13.8.22	feedbackPoint . . . . .	94
13.8.23	enableUseHalfFreqForContinuous . . . . .	94
<b>Chapter</b>	<b>DCP: Data Co-Processor</b>	
<b>14.1</b>	<b>Overview . . . . .</b>	<b>95</b>
<b>14.2</b>	<b>DCP Driver Initialization and Configuration . . . . .</b>	<b>95</b>
<b>14.3</b>	<b>Comments about API usage in RTOS . . . . .</b>	<b>96</b>
<b>14.4</b>	<b>Comments about API usage in interrupt handler . . . . .</b>	<b>96</b>
<b>14.5</b>	<b>DCP Driver Examples . . . . .</b>	<b>96</b>
14.5.1	Simple examples . . . . .	96
<b>14.6</b>	<b>Data Structure Documentation . . . . .</b>	<b>98</b>
14.6.1	struct dcp_work_packet_t . . . . .	98
14.6.2	struct dcp_handle_t . . . . .	98
14.6.3	struct dcp_context_t . . . . .	98
14.6.4	struct dcp_config_t . . . . .	98
<b>14.7</b>	<b>Macro Definition Documentation . . . . .</b>	<b>99</b>
14.7.1	FSL_DCP_DRIVER_VERSION . . . . .	99
<b>14.8</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>99</b>
14.8.1	_dcp_ch_enable_t . . . . .	99
14.8.2	_dcp_ch_int_enable_t . . . . .	99
14.8.3	dcp_channel_t . . . . .	99
14.8.4	dcp_key_slot_t . . . . .	100
<b>14.9</b>	<b>Function Documentation . . . . .</b>	<b>100</b>
14.9.1	DCP_Init . . . . .	100
14.9.2	DCP_Deinit . . . . .	100

# Contents

Section Number	Title	Page Number
14.9.3	DCP_GetDefaultConfig . . . . .	100
14.9.4	DCP_WaitForChannelComplete . . . . .	101
<b>14.10</b>	<b>DCP AES blocking driver . . . . .</b>	<b>102</b>
14.10.1	Overview . . . . .	102
14.10.2	Function Documentation . . . . .	102
<b>14.11</b>	<b>DCP AES non-blocking driver . . . . .</b>	<b>106</b>
14.11.1	Overview . . . . .	106
14.11.2	Function Documentation . . . . .	106
<b>14.12</b>	<b>DCP HASH driver . . . . .</b>	<b>110</b>
14.12.1	Overview . . . . .	110
14.12.2	Data Structure Documentation . . . . .	111
14.12.3	Macro Definition Documentation . . . . .	111
14.12.4	Enumeration Type Documentation . . . . .	111
14.12.5	Function Documentation . . . . .	111
<b>Chapter</b>	<b>DMAMUX: Direct Memory Access Multiplexer Driver</b>	
<b>15.1</b>	<b>Overview . . . . .</b>	<b>115</b>
<b>15.2</b>	<b>Typical use case . . . . .</b>	<b>115</b>
15.2.1	DMAMUX Operation . . . . .	115
<b>15.3</b>	<b>Macro Definition Documentation . . . . .</b>	<b>115</b>
15.3.1	FSL_DMAMUX_DRIVER_VERSION . . . . .	115
<b>15.4</b>	<b>Function Documentation . . . . .</b>	<b>116</b>
15.4.1	DMAMUX_Init . . . . .	116
15.4.2	DMAMUX_Deinit . . . . .	117
15.4.3	DMAMUX_EnableChannel . . . . .	117
15.4.4	DMAMUX_DisableChannel . . . . .	117
15.4.5	DMAMUX_SetSource . . . . .	118
15.4.6	DMAMUX_EnablePeriodTrigger . . . . .	118
15.4.7	DMAMUX_DisablePeriodTrigger . . . . .	118
15.4.8	DMAMUX_EnableAlwaysOn . . . . .	118
<b>Chapter</b>	<b>eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver</b>	
<b>16.1</b>	<b>Overview . . . . .</b>	<b>121</b>
<b>16.2</b>	<b>Typical use case . . . . .</b>	<b>121</b>
16.2.1	eDMA Operation . . . . .	121
<b>16.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>126</b>

# Contents

Section Number	Title	Page Number
16.3.1	struct edma_config_t . . . . .	126
16.3.2	struct edma_transfer_config_t . . . . .	127
16.3.3	struct edma_channel_Preemption_config_t . . . . .	127
16.3.4	struct edma_minor_offset_config_t . . . . .	128
16.3.5	struct edma_tcd_t . . . . .	128
16.3.6	struct edma_handle_t . . . . .	129
<b>16.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>130</b>
16.4.1	FSL_EDMA_DRIVER_VERSION . . . . .	130
<b>16.5</b>	<b>Typedef Documentation . . . . .</b>	<b>130</b>
16.5.1	edma_callback . . . . .	130
<b>16.6</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>131</b>
16.6.1	edma_transfer_size_t . . . . .	131
16.6.2	edma_modulo_t . . . . .	131
16.6.3	edma_bandwidth_t . . . . .	132
16.6.4	edma_channel_link_type_t . . . . .	132
16.6.5	_edma_channel_status_flags . . . . .	132
16.6.6	_edma_error_status_flags . . . . .	133
16.6.7	edma_interrupt_enable_t . . . . .	133
16.6.8	edma_transfer_type_t . . . . .	133
16.6.9	_edma_transfer_status . . . . .	133
<b>16.7</b>	<b>Function Documentation . . . . .</b>	<b>134</b>
16.7.1	EDMA_Init . . . . .	134
16.7.2	EDMA_Deinit . . . . .	135
16.7.3	EDMA_InstallTCD . . . . .	135
16.7.4	EDMA_GetDefaultConfig . . . . .	135
16.7.5	EDMA_ResetChannel . . . . .	136
16.7.6	EDMA_SetTransferConfig . . . . .	136
16.7.7	EDMA_SetMinorOffsetConfig . . . . .	137
16.7.8	EDMA_SetChannelPreemptionConfig . . . . .	137
16.7.9	EDMA_SetChannelLink . . . . .	137
16.7.10	EDMA_SetBandWidth . . . . .	139
16.7.11	EDMA_SetModulo . . . . .	140
16.7.12	EDMA_EnableAsyncRequest . . . . .	140
16.7.13	EDMA_EnableAutoStopRequest . . . . .	140
16.7.14	EDMA_EnableChannelInterrupts . . . . .	141
16.7.15	EDMA_DisableChannelInterrupts . . . . .	142
16.7.16	EDMA_TcdReset . . . . .	142
16.7.17	EDMA_TcdSetTransferConfig . . . . .	142
16.7.18	EDMA_TcdSetMinorOffsetConfig . . . . .	143
16.7.19	EDMA_TcdSetChannelLink . . . . .	143
16.7.20	EDMA_TcdSetBandWidth . . . . .	144

# Contents

Section Number	Title	Page Number
16.7.21	EDMA_TcdSetModulo . . . . .	144
16.7.22	EDMA_TcdEnableAutoStopRequest . . . . .	145
16.7.23	EDMA_TcdEnableInterrupts . . . . .	146
16.7.24	EDMA_TcdDisableInterrupts . . . . .	146
16.7.25	EDMA_EnableChannelRequest . . . . .	146
16.7.26	EDMA_DisableChannelRequest . . . . .	146
16.7.27	EDMA_TriggerChannelStart . . . . .	147
16.7.28	EDMA_GetRemainingMajorLoopCount . . . . .	147
16.7.29	EDMA_GetErrorStatusFlags . . . . .	148
16.7.30	EDMA_GetChannelStatusFlags . . . . .	148
16.7.31	EDMA_ClearChannelStatusFlags . . . . .	148
16.7.32	EDMA_CreateHandle . . . . .	149
16.7.33	EDMA_InstallTCDMemory . . . . .	150
16.7.34	EDMA_SetCallback . . . . .	150
16.7.35	EDMA_PrepareTransfer . . . . .	150
16.7.36	EDMA_SubmitTransfer . . . . .	151
16.7.37	EDMA_StartTransfer . . . . .	152
16.7.38	EDMA_StopTransfer . . . . .	152
16.7.39	EDMA_AbortTransfer . . . . .	152
16.7.40	EDMA_GetUnusedTCDDNumber . . . . .	152
16.7.41	EDMA_GetNextTCDDAddress . . . . .	153
16.7.42	EDMA_HandleIRQ . . . . .	153
<b>Chapter</b>	<b>eLCDIF: Enhanced LCD Interface</b>	
<b>17.1</b>	<b>Overview . . . . .</b>	<b>155</b>
<b>17.2</b>	<b>Typical use case . . . . .</b>	<b>155</b>
17.2.1	Frame buffer update . . . . .	155
17.2.2	Alpha surface . . . . .	155
<b>17.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>158</b>
17.3.1	struct elcdif_pixel_format_reg_t . . . . .	158
17.3.2	struct elcdif_rgb_mode_config_t . . . . .	158
17.3.3	struct elcdif_as_buffer_config_t . . . . .	159
17.3.4	struct elcdif_as_blend_config_t . . . . .	160
<b>17.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>160</b>
17.4.1	FSL_ELCDIF_DRIVER_VERSION . . . . .	160
<b>17.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>160</b>
17.5.1	_elcdif_polarity_flags . . . . .	160
17.5.2	_elcdif_interrupt_enable . . . . .	161
17.5.3	_elcdif_interrupt_flags . . . . .	161
17.5.4	_elcdif_status_flags . . . . .	161

# Contents

Section Number	Title	Page Number
17.5.5	elcdif_pixel_format_t . . . . .	161
17.5.6	elcdif_lcd_data_bus_t . . . . .	162
17.5.7	elcdif_as_pixel_format_t . . . . .	162
17.5.8	elcdif_alpha_mode_t . . . . .	162
17.5.9	elcdif_rop_mode_t . . . . .	163
17.5.10	elcdif_lut_t . . . . .	163
<b>17.6</b>	<b>Function Documentation . . . . .</b>	<b>163</b>
17.6.1	ELCDIF_RgbModeInit . . . . .	163
17.6.2	ELCDIF_GetStatus . . . . .	164
17.6.3	ELCDIF_GetLFifoCount . . . . .	165
17.6.4	ELCDIF_EnableInterrupts . . . . .	165
17.6.5	ELCDIF_DisableInterrupts . . . . .	166
17.6.6	ELCDIF_GetInterruptStatus . . . . .	166
17.6.7	ELCDIF_ClearInterruptStatus . . . . .	166
<b>Chapter</b>	<b>ENC: Quadrature Encoder/Decoder</b>	
<b>18.1</b>	<b>Overview . . . . .</b>	<b>167</b>
<b>18.2</b>	<b>Function groups . . . . .</b>	<b>167</b>
18.2.1	Initialization and De-initialization . . . . .	167
18.2.2	Status . . . . .	167
18.2.3	Interrupts . . . . .	167
18.2.4	Value Operation . . . . .	167
<b>18.3</b>	<b>Typical use case . . . . .</b>	<b>167</b>
18.3.1	Polling Configuration . . . . .	167
<b>18.4</b>	<b>Data Structure Documentation . . . . .</b>	<b>171</b>
18.4.1	struct enc_config_t . . . . .	171
18.4.2	struct enc_self_test_config_t . . . . .	172
<b>18.5</b>	<b>Macro Definition Documentation . . . . .</b>	<b>172</b>
18.5.1	FSL_ENC_DRIVER_VERSION . . . . .	172
<b>18.6</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>172</b>
18.6.1	_enc_interrupt_enable . . . . .	172
18.6.2	_enc_status_flags . . . . .	172
18.6.3	_enc_signal_status_flags . . . . .	173
18.6.4	enc_home_trigger_mode_t . . . . .	173
18.6.5	enc_index_trigger_mode_t . . . . .	173
18.6.6	enc_decoder_work_mode_t . . . . .	174
18.6.7	enc_position_match_mode_t . . . . .	174
18.6.8	enc_revolution_count_condition_t . . . . .	174
18.6.9	enc_self_test_direction_t . . . . .	174

# Contents

Section Number	Title	Page Number
<b>18.7</b>	<b>Function Documentation</b>	<b>175</b>
18.7.1	ENC_Init	175
18.7.2	ENC_Deinit	175
18.7.3	ENC_GetDefaultConfig	175
18.7.4	ENC_DoSoftwareLoadInitialPositionValue	176
18.7.5	ENC_SetSelfTestConfig	176
18.7.6	ENC_EnableWatchdog	176
18.7.7	ENC_SetInitialPositionValue	176
18.7.8	ENC_GetStatusFlags	177
18.7.9	ENC_ClearStatusFlags	177
18.7.10	ENC_GetSignalStatusFlags	177
18.7.11	ENC_EnableInterrupts	177
18.7.12	ENC_DisableInterrupts	178
18.7.13	ENC_GetEnabledInterrupts	178
18.7.14	ENC_GetPositionValue	178
18.7.15	ENC_GetHoldPositionValue	179
18.7.16	ENC_GetPositionDifferenceValue	180
18.7.17	ENC_GetHoldPositionDifferenceValue	180
18.7.18	ENC_GetRevolutionValue	180
18.7.19	ENC_GetHoldRevolutionValue	181
<b>18.8</b>	<b>Variable Documentation</b>	<b>181</b>
18.8.1	enableReverseDirection	181
18.8.2	decoderWorkMode	181
18.8.3	HOMETriggerMode	181
18.8.4	INDEXTriggerMode	181
18.8.5	enableTRIGGERClearPositionCounter	181
18.8.6	enableWatchdog	181
18.8.7	watchdogTimeoutValue	181
18.8.8	filterCount	182
18.8.9	filterSamplePeriod	182
18.8.10	positionMatchMode	182
18.8.11	positionCompareValue	182
18.8.12	revolutionCountCondition	182
18.8.13	enableModuloCountMode	182
18.8.14	positionModulusValue	182
18.8.15	positionInitialValue	182
18.8.16	signalDirection	182
18.8.17	signalCount	182
18.8.18	signalPeriod	183
<b>Chapter</b>	<b>ENET: Ethernet MAC Driver</b>	
<b>19.1</b>	<b>Overview</b>	<b>185</b>

# Contents

Section Number	Title	Page Number
<b>19.2</b>	<b>Typical use case</b>	<b>186</b>
19.2.1	ENET Initialization, receive, and transmit operations	186
<b>19.3</b>	<b>Data Structure Documentation</b>	<b>194</b>
19.3.1	struct enet_rx_bd_struct_t	194
19.3.2	struct enet_tx_bd_struct_t	195
19.3.3	struct enet_data_error_stats_t	196
19.3.4	struct enet_buffer_config_t	197
19.3.5	struct enet_ptp_time_t	198
19.3.6	struct enet_ptp_time_data_t	199
19.3.7	struct enet_ptp_time_data_ring_t	199
19.3.8	struct enet_ptp_config_t	200
19.3.9	struct enet_intcoalesce_config_t	200
19.3.10	struct enet_config_t	201
19.3.11	struct _enet_handle	203
<b>19.4</b>	<b>Macro Definition Documentation</b>	<b>204</b>
19.4.1	FSL_ENET_DRIVER_VERSION	204
19.4.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	206
19.4.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	206
19.4.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	206
19.4.5	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	206
19.4.6	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	206
19.4.7	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	206
19.4.8	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	206
19.4.9	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	206
19.4.10	ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK	206
19.4.11	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	206
19.4.12	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	206
19.4.13	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	206
19.4.14	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	206
19.4.15	ENET_BUFFDESCRIPTOR_TX_READY_MASK	206
19.4.16	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK	206
19.4.17	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	206
19.4.18	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK	206
19.4.19	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	206
19.4.20	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	206
19.4.21	ENET_BUFFDESCRIPTOR_RX_IPV4_MASK	206
19.4.22	ENET_BUFFDESCRIPTOR_RX_IPV6_MASK	206
19.4.23	ENET_BUFFDESCRIPTOR_RX_VLAN_MASK	206
19.4.24	ENET_BUFFDESCRIPTOR_RX_PROTOCOLCHECKSUM_MASK	206
19.4.25	ENET_BUFFDESCRIPTOR_RX_IPHEADCHECKSUM_MASK	206
19.4.26	ENET_BUFFDESCRIPTOR_RX_INTERRUPT_MASK	206
19.4.27	ENET_BUFFDESCRIPTOR_RX_UNICAST_MASK	206
19.4.28	ENET_BUFFDESCRIPTOR_RX_COLLISION_MASK	206



# Contents

Section Number	Title	Page Number
19.4.29	ENET_BUFFDESCRIPTOR_RX_PHYERR_MASK	206
19.4.30	ENET_BUFFDESCRIPTOR_RX_MACERR_MASK	206
19.4.31	ENET_BUFFDESCRIPTOR_TX_ERR_MASK	206
19.4.32	ENET_BUFFDESCRIPTOR_TX_UNDERFLOWERR_MASK	206
19.4.33	ENET_BUFFDESCRIPTOR_TX_EXCCOLLISIONERR_MASK	206
19.4.34	ENET_BUFFDESCRIPTOR_TX_FRAMEERR_MASK	206
19.4.35	ENET_BUFFDESCRIPTOR_TX_LATECOLLISIONERR_MASK	206
19.4.36	ENET_BUFFDESCRIPTOR_TX_OVERFLOWERR_MASK	206
19.4.37	ENET_BUFFDESCRIPTOR_TX_TIMESTAMPERR_MASK	206
19.4.38	ENET_BUFFDESCRIPTOR_TX_INTERRUPT_MASK	206
19.4.39	ENET_BUFFDESCRIPTOR_TX_TIMESTAMP_MASK	206
19.4.40	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	206
19.4.41	ENET_FRAME_MAX_FRAMELEN	207
19.4.42	ENET_FIFO_MIN_RX_FULL	207
19.4.43	ENET_RX_MIN_BUFFERSIZE	207
<b>19.5</b>	<b>Typedef Documentation</b>	<b>207</b>
19.5.1	enet_callback_t	207
<b>19.6</b>	<b>Enumeration Type Documentation</b>	<b>207</b>
19.6.1	_enet_status	207
19.6.2	enet_mii_mode_t	207
19.6.3	enet_mii_speed_t	207
19.6.4	enet_mii_duplex_t	208
19.6.5	enet_mii_write_t	208
19.6.6	enet_mii_read_t	208
19.6.7	enet_mii_extend_opcode	208
19.6.8	enet_special_control_flag_t	208
19.6.9	enet_interrupt_enable_t	209
19.6.10	enet_event_t	209
19.6.11	enet_tx_accelerator_t	210
19.6.12	enet_rx_accelerator_t	210
19.6.13	enet_ptp_event_type_t	210
19.6.14	enet_ptp_timer_channel_t	210
19.6.15	enet_ptp_timer_channel_mode_t	211
<b>19.7</b>	<b>Function Documentation</b>	<b>211</b>
19.7.1	ENET_GetDefaultConfig	211
19.7.2	ENET_Init	211
19.7.3	ENET_Deinit	212
19.7.4	ENET_Reset	212
19.7.5	ENET_SetMII	213
19.7.6	ENET_SetSMI	213
19.7.7	ENET_GetSMI	213
19.7.8	ENET_ReadSMIData	214

# Contents

<b>Section Number</b>	<b>Title</b>	<b>Page Number</b>
19.7.9	ENET_StartSMIRead . . . . .	214
19.7.10	ENET_StartSMIWrite . . . . .	214
19.7.11	ENET_StartExtC45SMIRead . . . . .	215
19.7.12	ENET_StartExtC45SMIWrite . . . . .	215
19.7.13	ENET_SetMacAddr . . . . .	215
19.7.14	ENET_GetMacAddr . . . . .	216
19.7.15	ENET_AddMulticastGroup . . . . .	216
19.7.16	ENET_LeaveMulticastGroup . . . . .	216
19.7.17	ENET_ActiveRead . . . . .	216
19.7.18	ENET_EnableSleepMode . . . . .	217
19.7.19	ENET_GetAccelFunction . . . . .	217
19.7.20	ENET_EnableInterrupts . . . . .	217
19.7.21	ENET_DisableInterrupts . . . . .	219
19.7.22	ENET_GetInterruptStatus . . . . .	219
19.7.23	ENET_ClearInterruptStatus . . . . .	219
19.7.24	ENET_SetCallback . . . . .	220
19.7.25	ENET_GetRxErrBeforeReadFrame . . . . .	220
19.7.26	ENET_GetTxErrAfterSendFrame . . . . .	221
19.7.27	ENET_GetRxFrameSize . . . . .	221
19.7.28	ENET_ReadFrame . . . . .	222
19.7.29	ENET_SendFrame . . . . .	223
19.7.30	ENET_TransmitIRQHandler . . . . .	223
19.7.31	ENET_ReceiveIRQHandler . . . . .	224
19.7.32	ENET_ErrorIRQHandler . . . . .	224
19.7.33	ENET_CommonFrame0IRQHandler . . . . .	224
19.7.34	ENET_Ptp1588Configure . . . . .	224
19.7.35	ENET_Ptp1588StartTimer . . . . .	225
19.7.36	ENET_Ptp1588StopTimer . . . . .	225
19.7.37	ENET_Ptp1588AdjustTimer . . . . .	226
19.7.38	ENET_Ptp1588SetChannelMode . . . . .	226
19.7.39	ENET_Ptp1588SetChannelOutputPulseWidth . . . . .	226
19.7.40	ENET_Ptp1588SetChannelCmpValue . . . . .	227
19.7.41	ENET_Ptp1588GetChannelStatus . . . . .	227
19.7.42	ENET_Ptp1588ClearChannelStatus . . . . .	227
19.7.43	ENET_Ptp1588GetTimer . . . . .	228
19.7.44	ENET_Ptp1588SetTimer . . . . .	228
19.7.45	ENET_Ptp1588TimerIRQHandler . . . . .	228
19.7.46	ENET_GetRxFrameTime . . . . .	228
19.7.47	ENET_GetTxFrameTime . . . . .	229
<b>Chapter</b>	<b>EWM: External Watchdog Monitor Driver</b>	
<b>20.1</b>	<b>Overview . . . . .</b>	<b>231</b>
<b>20.2</b>	<b>Typical use case . . . . .</b>	<b>231</b>

# Contents

Section Number	Title	Page Number
<b>20.3</b>	<b>Data Structure Documentation</b>	<b>232</b>
20.3.1	struct ewm_config_t	232
<b>20.4</b>	<b>Macro Definition Documentation</b>	<b>232</b>
20.4.1	FSL_EWM_DRIVER_VERSION	232
<b>20.5</b>	<b>Enumeration Type Documentation</b>	<b>232</b>
20.5.1	_ewm_interrupt_enable_t	232
20.5.2	_ewm_status_flags_t	232
<b>20.6</b>	<b>Function Documentation</b>	<b>233</b>
20.6.1	EWM_Init	233
20.6.2	EWM_Deinit	233
20.6.3	EWM_GetDefaultConfig	233
20.6.4	EWM_EnableInterrupts	234
20.6.5	EWM_DisableInterrupts	234
20.6.6	EWM_GetStatusFlags	234
20.6.7	EWM_Refresh	235
<b>Chapter FlexCAN: Flex Controller Area Network Driver</b>		
<b>21.1</b>	<b>Overview</b>	<b>237</b>
<b>21.2</b>	<b>FlexCAN Driver</b>	<b>238</b>
21.2.1	Overview	238
21.2.2	Typical use case	238
21.2.3	Data Structure Documentation	245
21.2.4	Macro Definition Documentation	249
21.2.5	Typedef Documentation	254
21.2.6	Enumeration Type Documentation	254
21.2.7	Function Documentation	257
<b>21.3</b>	<b>FlexCAN eDMA Driver</b>	<b>271</b>
21.3.1	Overview	271
21.3.2	Data Structure Documentation	271
21.3.3	Macro Definition Documentation	272
21.3.4	Typedef Documentation	272
21.3.5	Function Documentation	272
<b>Chapter FLEXRAM: on-chip RAM manager</b>		
<b>22.1</b>	<b>Overview</b>	<b>275</b>
<b>22.2</b>	<b>Data Structure Documentation</b>	<b>277</b>
22.2.1	struct flexram_allocate_ram_t	277

# Contents

Section Number	Title	Page Number
<b>22.3</b>	<b>Macro Definition Documentation</b>	<b>278</b>
22.3.1	FSL_FLEXRAM_DRIVER_VERSION	278
<b>22.4</b>	<b>Enumeration Type Documentation</b>	<b>278</b>
22.4.1	_flexram_wr_rd_sel	278
22.4.2	_flexram_interrupt_status	278
22.4.3	flexram_tcm_access_mode_t	278
22.4.4	_flexram_bank_type	278
22.4.5	_flexram_tcm_size	279
22.4.6	flexram_bank_allocate_src_t	279
<b>22.5</b>	<b>Function Documentation</b>	<b>279</b>
22.5.1	FLEXRAM_Init	279
22.5.2	FLEXRAM_GetInterruptStatus	279
22.5.3	FLEXRAM_ClearInterruptStatus	279
22.5.4	FLEXRAM_EnableInterruptStatus	280
22.5.5	FLEXRAM_DisableInterruptStatus	280
22.5.6	FLEXRAM_EnableInterruptSignal	280
22.5.7	FLEXRAM_DisableInterruptSignal	280
22.5.8	FLEXRAM_SetTCMReadAccessMode	281
22.5.9	FLEXRAM_SetTCMWriteAccessMode	281
22.5.10	FLEXRAM_EnableForceRamClockOn	281
22.5.11	FLEXRAM_SetOCRAMMagicAddr	281
22.5.12	FLEXRAM_SetDTCMMagicAddr	282
22.5.13	FLEXRAM_SetITCMMagicAddr	282
22.5.14	FLEXRAM_AllocateRam	282
22.5.15	FLEXRAM_SetAllocateRamSrc	282
<b>Chapter</b>	<b>FlexIO: FlexIO Driver</b>	
<b>23.1</b>	<b>Overview</b>	<b>285</b>
<b>23.2</b>	<b>FlexIO Driver</b>	<b>286</b>
23.2.1	Overview	286
23.2.2	Data Structure Documentation	290
23.2.3	Macro Definition Documentation	293
23.2.4	Typedef Documentation	293
23.2.5	Enumeration Type Documentation	293
23.2.6	Function Documentation	297
23.2.7	Variable Documentation	307
<b>23.3</b>	<b>FlexIO Camera Driver</b>	<b>308</b>
23.3.1	Overview	308
23.3.2	Typical use case	308
23.3.3	Data Structure Documentation	311

# Contents

Section Number	Title	Page Number
23.3.4	Macro Definition Documentation . . . . .	312
23.3.5	Enumeration Type Documentation . . . . .	312
23.3.6	Function Documentation . . . . .	313
23.3.7	FlexIO eDMA Camera Driver . . . . .	317
<b>23.4</b>	<b>FlexIO I2C Master Driver . . . . .</b>	<b>321</b>
23.4.1	Overview . . . . .	321
23.4.2	Typical use case . . . . .	321
23.4.3	Data Structure Documentation . . . . .	325
23.4.4	Macro Definition Documentation . . . . .	328
23.4.5	Typedef Documentation . . . . .	328
23.4.6	Enumeration Type Documentation . . . . .	328
23.4.7	Function Documentation . . . . .	329
<b>23.5</b>	<b>FlexIO I2S Driver . . . . .</b>	<b>339</b>
23.5.1	Overview . . . . .	339
23.5.2	Typical use case . . . . .	339
23.5.3	Data Structure Documentation . . . . .	344
23.5.4	Macro Definition Documentation . . . . .	346
23.5.5	Enumeration Type Documentation . . . . .	346
23.5.6	Function Documentation . . . . .	348
23.5.7	FlexIO eDMA I2S Driver . . . . .	359
23.5.8	FlexIO DMA I2S Driver . . . . .	366
<b>23.6</b>	<b>FlexIO MCU Interface LCD Driver . . . . .</b>	<b>373</b>
23.6.1	Overview . . . . .	373
23.6.2	Typical use case . . . . .	373
23.6.3	Data Structure Documentation . . . . .	379
23.6.4	Macro Definition Documentation . . . . .	382
23.6.5	Typedef Documentation . . . . .	383
23.6.6	Enumeration Type Documentation . . . . .	383
23.6.7	Function Documentation . . . . .	384
23.6.8	FlexIO eDMA MCU Interface LCD Driver . . . . .	397
<b>23.7</b>	<b>FlexIO SPI Driver . . . . .</b>	<b>399</b>
23.7.1	Overview . . . . .	399
23.7.2	Typical use case . . . . .	399
23.7.3	Data Structure Documentation . . . . .	406
23.7.4	Macro Definition Documentation . . . . .	410
23.7.5	Typedef Documentation . . . . .	410
23.7.6	Enumeration Type Documentation . . . . .	410
23.7.7	Function Documentation . . . . .	412
23.7.8	FlexIO eDMA SPI Driver . . . . .	426
23.7.9	FlexIO DMA SPI Driver . . . . .	434

# Contents

Section Number	Title	Page Number
<b>23.8</b>	<b>FlexIO UART Driver</b>	<b>442</b>
23.8.1	Overview	442
23.8.2	Typical use case	442
23.8.3	Data Structure Documentation	450
23.8.4	Macro Definition Documentation	453
23.8.5	Typedef Documentation	453
23.8.6	Enumeration Type Documentation	453
23.8.7	Function Documentation	454
23.8.8	FlexIO eDMA UART Driver	467
23.8.9	FlexIO DMA UART Driver	473
<b>Chapter</b>	<b>FLEXSPI: Flexible Serial Peripheral Interface Driver</b>	
<b>24.1</b>	<b>Overview</b>	<b>479</b>
<b>24.2</b>	<b>Data Structure Documentation</b>	<b>484</b>
24.2.1	struct flexspi_config_t	484
24.2.2	struct flexspi_device_config_t	487
24.2.3	struct flexspi_transfer_t	488
24.2.4	struct _flexspi_handle	489
<b>24.3</b>	<b>Macro Definition Documentation</b>	<b>490</b>
24.3.1	FSL_FLEXSPI_DRIVER_VERSION	490
24.3.2	FLEXSPI_LUT_SEQ	490
<b>24.4</b>	<b>Typedef Documentation</b>	<b>490</b>
24.4.1	flexspi_transfer_callback_t	490
<b>24.5</b>	<b>Enumeration Type Documentation</b>	<b>490</b>
24.5.1	_flexspi_status	490
24.5.2	_flexspi_command	490
24.5.3	_flexspi_pad	491
24.5.4	flexspi_flags_t	491
24.5.5	flexspi_read_sample_clock_t	492
24.5.6	flexspi_cs_interval_cycle_unit_t	492
24.5.7	flexspi_ahb_write_wait_unit_t	492
24.5.8	flexspi_ip_error_code_t	493
24.5.9	flexspi_ahb_error_code_t	493
24.5.10	flexspi_port_t	493
24.5.11	flexspi_arb_command_source_t	494
24.5.12	flexspi_command_type_t	494
<b>24.6</b>	<b>Function Documentation</b>	<b>494</b>
24.6.1	FLEXSPI_Init	494
24.6.2	FLEXSPI_GetDefaultConfig	494

# Contents

Section Number	Title	Page Number
24.6.3	FLEXSPI_Deinit . . . . .	494
24.6.4	FLEXSPI_SetFlashConfig . . . . .	495
24.6.5	FLEXSPI_SoftwareReset . . . . .	495
24.6.6	FLEXSPI_Enable . . . . .	495
24.6.7	FLEXSPI_EnableInterrupts . . . . .	495
24.6.8	FLEXSPI_DisableInterrupts . . . . .	496
24.6.9	FLEXSPI_EnableTxDMA . . . . .	496
24.6.10	FLEXSPI_EnableRxDMA . . . . .	496
24.6.11	FLEXSPI_GetTxFifoAddress . . . . .	496
24.6.12	FLEXSPI_GetRxFifoAddress . . . . .	497
24.6.13	FLEXSPI_ResetFifos . . . . .	497
24.6.14	FLEXSPI_GetFifoCounts . . . . .	497
24.6.15	FLEXSPI_GetInterruptStatusFlags . . . . .	498
24.6.16	FLEXSPI_ClearInterruptStatusFlags . . . . .	498
24.6.17	FLEXSPI_GetArbitratorCommandSource . . . . .	498
24.6.18	FLEXSPI_GetIPCommandErrorCode . . . . .	498
24.6.19	FLEXSPI_GetAHBCommandErrorCode . . . . .	499
24.6.20	FLEXSPI_GetBusIdleStatus . . . . .	499
24.6.21	FLEXSPI_EnableIPParallelMode . . . . .	499
24.6.22	FLEXSPI_EnableAHBParallelMode . . . . .	500
24.6.23	FLEXSPI_UpdateLUT . . . . .	500
24.6.24	FLEXSPI_WriteData . . . . .	500
24.6.25	FLEXSPI_ReadData . . . . .	501
24.6.26	FLEXSPI_WriteBlocking . . . . .	502
24.6.27	FLEXSPI_ReadBlocking . . . . .	502
24.6.28	FLEXSPI_TransferBlocking . . . . .	503
24.6.29	FLEXSPI_TransferCreateHandle . . . . .	504
24.6.30	FLEXSPI_TransferNonBlocking . . . . .	505
24.6.31	FLEXSPI_TransferGetCount . . . . .	505
24.6.32	FLEXSPI_TransferAbort . . . . .	506
24.6.33	FLEXSPI_TransferHandleIRQ . . . . .	506
<b>24.7</b>	<b>FLEXSPI eDMA Driver . . . . .</b>	<b>507</b>
24.7.1	Overview . . . . .	507
24.7.2	Data Structure Documentation . . . . .	508
24.7.3	Macro Definition Documentation . . . . .	509
24.7.4	Enumeration Type Documentation . . . . .	509
24.7.5	Function Documentation . . . . .	509
<b>Chapter</b>	<b>GPC: General Power Controller Driver</b>	
<b>25.1</b>	<b>Overview . . . . .</b>	<b>513</b>
<b>25.2</b>	<b>Typical use case . . . . .</b>	<b>513</b>

# Contents

Section Number	Title	Page Number
<b>25.3</b>	<b>Macro Definition Documentation</b>	<b>514</b>
25.3.1	FSL_GPC_DRIVER_VERSION	514
<b>25.4</b>	<b>Enumeration Type Documentation</b>	<b>514</b>
25.4.1	_gpc_memory_power_gate	514
25.4.2	_gpc_status_flags	514
<b>25.5</b>	<b>Function Documentation</b>	<b>515</b>
25.5.1	GPC_EnableMemoryGate	515
25.5.2	GPC_EnablePartialSleepWakeupSource	516
25.5.3	GPC_GetPartialSleepWakeupFlag	516
25.5.4	GPC_EnablePartialSleepMode	517
25.5.5	GPC_ConfigPowerUpSequence	517
25.5.6	GPC_ConfigPowerDownSequence	517
25.5.7	GPC_GetStatusFlags	518
25.5.8	GPC_ClearStatusFlags	518
<b>Chapter</b>	<b>GPIO: General-Purpose Input/Output Driver</b>	
<b>26.1</b>	<b>Overview</b>	<b>519</b>
<b>26.2</b>	<b>GPIO Driver</b>	<b>520</b>
26.2.1	Overview	520
26.2.2	Typical use case	520
26.2.3	Data Structure Documentation	522
26.2.4	Macro Definition Documentation	522
26.2.5	Enumeration Type Documentation	522
26.2.6	Function Documentation	523
<b>Chapter</b>	<b>GPT: General Purpose Timer</b>	
<b>27.1</b>	<b>Overview</b>	<b>529</b>
<b>27.2</b>	<b>Function groups</b>	<b>529</b>
27.2.1	Initialization and deinitialization	529
<b>27.3</b>	<b>Typical use case</b>	<b>529</b>
27.3.1	GPT interrupt example	529
<b>27.4</b>	<b>Data Structure Documentation</b>	<b>532</b>
27.4.1	struct gpt_config_t	532
<b>27.5</b>	<b>Enumeration Type Documentation</b>	<b>533</b>
27.5.1	gpt_clock_source_t	533
27.5.2	gpt_input_capture_channel_t	533
27.5.3	gpt_input_operation_mode_t	534



# Contents

Section Number	Title	Page Number
27.5.4	<code>gpt_output_compare_channel_t</code> . . . . .	534
27.5.5	<code>gpt_output_operation_mode_t</code> . . . . .	534
27.5.6	<code>gpt_interrupt_enable_t</code> . . . . .	534
27.5.7	<code>gpt_status_flag_t</code> . . . . .	535
<b>27.6</b>	<b>Function Documentation</b> . . . . .	<b>535</b>
27.6.1	<code>GPT_Init</code> . . . . .	535
27.6.2	<code>GPT_Deinit</code> . . . . .	535
27.6.3	<code>GPT_GetDefaultConfig</code> . . . . .	535
27.6.4	<code>GPT_SoftwareReset</code> . . . . .	536
27.6.5	<code>GPT_SetClockSource</code> . . . . .	536
27.6.6	<code>GPT_GetClockSource</code> . . . . .	536
27.6.7	<code>GPT_SetClockDivider</code> . . . . .	536
27.6.8	<code>GPT_GetClockDivider</code> . . . . .	537
27.6.9	<code>GPT_SetOscClockDivider</code> . . . . .	537
27.6.10	<code>GPT_GetOscClockDivider</code> . . . . .	537
27.6.11	<code>GPT_StartTimer</code> . . . . .	537
27.6.12	<code>GPT_StopTimer</code> . . . . .	538
27.6.13	<code>GPT_GetCurrentTimerCount</code> . . . . .	538
27.6.14	<code>GPT_SetInputOperationMode</code> . . . . .	538
27.6.15	<code>GPT_GetInputOperationMode</code> . . . . .	538
27.6.16	<code>GPT_GetInputCaptureValue</code> . . . . .	539
27.6.17	<code>GPT_SetOutputOperationMode</code> . . . . .	539
27.6.18	<code>GPT_GetOutputOperationMode</code> . . . . .	540
27.6.19	<code>GPT_SetOutputCompareValue</code> . . . . .	540
27.6.20	<code>GPT_GetOutputCompareValue</code> . . . . .	540
27.6.21	<code>GPT_ForceOutput</code> . . . . .	541
27.6.22	<code>GPT_EnableInterrupts</code> . . . . .	541
27.6.23	<code>GPT_DisableInterrupts</code> . . . . .	541
27.6.24	<code>GPT_GetEnabledInterrupts</code> . . . . .	541
27.6.25	<code>GPT_GetStatusFlags</code> . . . . .	542
27.6.26	<code>GPT_ClearStatusFlags</code> . . . . .	542
<b>Chapter</b>	<b>KPP: KeyPad Port Driver</b>	
<b>28.1</b>	<b>Overview</b> . . . . .	<b>543</b>
<b>28.2</b>	<b>Typical use case</b> . . . . .	<b>543</b>
<b>28.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>544</b>
28.3.1	<code>struct kpp_config_t</code> . . . . .	544
<b>28.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>544</b>
28.4.1	<code>FSL_KPP_DRIVER_VERSION</code> . . . . .	544

# Contents

Section Number	Title	Page Number
<b>28.5</b>	<b>Enumeration Type Documentation</b>	<b>544</b>
28.5.1	kpp_interrupt_enable_t	544
28.5.2	kpp_sync_operation_t	545
<b>28.6</b>	<b>Function Documentation</b>	<b>545</b>
28.6.1	KPP_Init	545
28.6.2	KPP_Deinit	545
28.6.3	KPP_EnableInterrupts	545
28.6.4	KPP_DisableInterrupts	546
28.6.5	KPP_GetStatusFlag	546
28.6.6	KPP_ClearStatusFlag	546
28.6.7	KPP_SetSynchronizeChain	546
28.6.8	KPP_keyPressScanning	547
<b>Chapter</b>	<b>LPI2C: Low Power I2C Driver</b>	
<b>29.1</b>	<b>Overview</b>	<b>549</b>
<b>29.2</b>	<b>Macro Definition Documentation</b>	<b>549</b>
29.2.1	FSL_LPI2C_DRIVER_VERSION	549
29.2.2	LPI2C_WAIT_TIMEOUT	549
<b>29.3</b>	<b>Enumeration Type Documentation</b>	<b>550</b>
29.3.1	_lpi2c_status	550
<b>29.4</b>	<b>LPI2C Master Driver</b>	<b>551</b>
29.4.1	Overview	551
29.4.2	Data Structure Documentation	554
29.4.3	Typedef Documentation	558
29.4.4	Enumeration Type Documentation	558
29.4.5	Function Documentation	561
<b>29.5</b>	<b>LPI2C Slave Driver</b>	<b>575</b>
29.5.1	Overview	575
29.5.2	Data Structure Documentation	577
29.5.3	Typedef Documentation	581
29.5.4	Enumeration Type Documentation	582
29.5.5	Function Documentation	583
<b>29.6</b>	<b>LPI2C Master DMA Driver</b>	<b>592</b>
29.6.1	Overview	592
29.6.2	Data Structure Documentation	592
29.6.3	Typedef Documentation	594
29.6.4	Function Documentation	595
<b>29.7</b>	<b>LPI2C FreeRTOS Driver</b>	<b>598</b>

# Contents

Section Number	Title	Page Number
29.7.1	Overview . . . . .	598
29.7.2	Macro Definition Documentation . . . . .	598
29.7.3	Function Documentation . . . . .	598
<b>Chapter LPSPI: Low Power Serial Peripheral Interface</b>		
<b>30.1</b>	<b>Overview . . . . .</b>	<b>601</b>
<b>30.2</b>	<b>LPSPI Peripheral driver . . . . .</b>	<b>602</b>
30.2.1	Overview . . . . .	602
30.2.2	Function groups . . . . .	602
30.2.3	Typical use case . . . . .	602
30.2.4	Data Structure Documentation . . . . .	609
30.2.5	Macro Definition Documentation . . . . .	615
30.2.6	Typedef Documentation . . . . .	616
30.2.7	Enumeration Type Documentation . . . . .	617
30.2.8	Function Documentation . . . . .	622
30.2.9	Variable Documentation . . . . .	638
<b>30.3</b>	<b>LPSPI eDMA Driver . . . . .</b>	<b>639</b>
30.3.1	Overview . . . . .	639
30.3.2	Data Structure Documentation . . . . .	640
30.3.3	Macro Definition Documentation . . . . .	645
30.3.4	Typedef Documentation . . . . .	645
30.3.5	Function Documentation . . . . .	646
<b>30.4</b>	<b>LPSPI FreeRTOS Driver . . . . .</b>	<b>651</b>
30.4.1	Overview . . . . .	651
30.4.2	Macro Definition Documentation . . . . .	651
30.4.3	Function Documentation . . . . .	651
<b>Chapter LPUART: Low Power UART Driver</b>		
<b>31.1</b>	<b>Overview . . . . .</b>	<b>655</b>
<b>31.2</b>	<b>LPUART Driver . . . . .</b>	<b>656</b>
31.2.1	Overview . . . . .	656
31.2.2	Typical use case . . . . .	656
31.2.3	Data Structure Documentation . . . . .	661
31.2.4	Macro Definition Documentation . . . . .	664
31.2.5	Typedef Documentation . . . . .	664
31.2.6	Enumeration Type Documentation . . . . .	664
31.2.7	Function Documentation . . . . .	668
<b>31.3</b>	<b>LPUART DMA Driver . . . . .</b>	<b>683</b>

# Contents

Section Number	Title	Page Number
31.3.1	Overview . . . . .	683
31.3.2	Data Structure Documentation . . . . .	684
31.3.3	Macro Definition Documentation . . . . .	685
31.3.4	Typedef Documentation . . . . .	685
31.3.5	Function Documentation . . . . .	685
<b>31.4</b>	<b>LPUART eDMA Driver . . . . .</b>	<b>689</b>
31.4.1	Overview . . . . .	689
31.4.2	Data Structure Documentation . . . . .	690
31.4.3	Macro Definition Documentation . . . . .	691
31.4.4	Typedef Documentation . . . . .	691
31.4.5	Function Documentation . . . . .	691
<b>31.5</b>	<b>LPUART FreeRTOS Driver . . . . .</b>	<b>695</b>
31.5.1	Overview . . . . .	695
31.5.2	Data Structure Documentation . . . . .	695
31.5.3	Macro Definition Documentation . . . . .	696
31.5.4	Function Documentation . . . . .	696
<b>Chapter</b>	<b>PIT: Periodic Interrupt Timer</b>	
<b>32.1</b>	<b>Overview . . . . .</b>	<b>699</b>
<b>32.2</b>	<b>Function groups . . . . .</b>	<b>699</b>
32.2.1	Initialization and deinitialization . . . . .	699
32.2.2	Timer period Operations . . . . .	699
32.2.3	Start and Stop timer operations . . . . .	699
32.2.4	Status . . . . .	700
32.2.5	Interrupt . . . . .	700
<b>32.3</b>	<b>Typical use case . . . . .</b>	<b>700</b>
32.3.1	PIT tick example . . . . .	700
<b>32.4</b>	<b>Data Structure Documentation . . . . .</b>	<b>701</b>
32.4.1	struct pit_config_t . . . . .	701
<b>32.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>702</b>
32.5.1	pit_chnl_t . . . . .	702
32.5.2	pit_interrupt_enable_t . . . . .	702
32.5.3	pit_status_flags_t . . . . .	702
<b>32.6</b>	<b>Function Documentation . . . . .</b>	<b>702</b>
32.6.1	PIT_Init . . . . .	702
32.6.2	PIT_Deinit . . . . .	703
32.6.3	PIT_GetDefaultConfig . . . . .	703
32.6.4	PIT_SetTimerChainMode . . . . .	703

# Contents

Section Number	Title	Page Number
32.6.5	PIT_EnableInterrupts . . . . .	704
32.6.6	PIT_DisableInterrupts . . . . .	704
32.6.7	PIT_GetEnabledInterrupts . . . . .	704
32.6.8	PIT_GetStatusFlags . . . . .	705
32.6.9	PIT_ClearStatusFlags . . . . .	706
32.6.10	PIT_SetTimerPeriod . . . . .	706
32.6.11	PIT_GetCurrentTimerCount . . . . .	707
32.6.12	PIT_StartTimer . . . . .	707
32.6.13	PIT_StopTimer . . . . .	707
32.6.14	PIT_GetLifetimeTimerCount . . . . .	708
<b>Chapter PMU: Power Management Unit</b>		
<b>33.1</b>	<b>Overview . . . . .</b>	<b>709</b>
<b>33.2</b>	<b>Macro Definition Documentation . . . . .</b>	<b>711</b>
33.2.1	FSL_PMU_DRIVER_VERSION . . . . .	711
<b>33.3</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>711</b>
33.3.1	_pmu_status_flags . . . . .	711
33.3.2	pmu_1p1_weak_reference_source_t . . . . .	712
33.3.3	pmu_3p0_vbus_voltage_source_t . . . . .	712
33.3.4	pmu_core_reg_voltage_ramp_rate_t . . . . .	712
33.3.5	pmu_power_bandgap_t . . . . .	712
<b>33.4</b>	<b>Function Documentation . . . . .</b>	<b>712</b>
33.4.1	PMU_1P1SetWeakReferenceSource . . . . .	712
33.4.2	PMU_1P1EnableWeakRegulator . . . . .	713
33.4.3	PMU_1P1SetRegulatorOutputVoltage . . . . .	713
33.4.4	PMU_1P1SetBrownoutOffsetVoltage . . . . .	713
33.4.5	PMU_1P1EnablePullDown . . . . .	714
33.4.6	PMU_1P1EnableCurrentLimit . . . . .	714
33.4.7	PMU_1P1EnableBrownout . . . . .	714
33.4.8	PMU_1P1EnableOutput . . . . .	714
33.4.9	PMU_3P0SetRegulatorOutputVoltage . . . . .	715
33.4.10	PMU_3P0SetVBusVoltageSource . . . . .	715
33.4.11	PMU_3P0SetBrownoutOffsetVoltage . . . . .	715
33.4.12	PMU_3P0EnableCurrentLimit . . . . .	716
33.4.13	PMU_3P0EnableBrownout . . . . .	716
33.4.14	PMU_3P0EnableOutput . . . . .	716
33.4.15	PMU_2P5EnableWeakRegulator . . . . .	716
33.4.16	PMU_2P5SetRegulatorOutputVoltage . . . . .	717
33.4.17	PMU_2P5SetBrownoutOffsetVoltage . . . . .	717
33.4.18	PMU_2P5EnablePullDown . . . . .	717
33.4.19	PMU_2P1EnablePullDown . . . . .	718

# Contents

Section Number	Title	Page Number
33.4.20	PMU_2P5EnableCurrentLimit	718
33.4.21	PMU_2P5nableBrownout	718
33.4.22	PMU_2P5EnableOutput	718
33.4.23	PMU_CoreEnableIncreaseGateDrive	719
33.4.24	PMU_CoreSetRegulatorVoltageRampRate	719
33.4.25	PMU_CoreSetSOCDomainVoltage	719
33.4.26	PMU_CoreSetARMCoreDomainVoltage	720
<b>Chapter PWM: Pulse Width Modulator</b>		
<b>34.1</b>	<b>Overview</b>	<b>721</b>
<b>34.2</b>	<b>Register Update</b>	<b>721</b>
<b>34.3</b>	<b>Typical use case</b>	<b>722</b>
34.3.1	PWM output	722
<b>34.4</b>	<b>Data Structure Documentation</b>	<b>728</b>
34.4.1	struct pwm_signal_param_t	728
34.4.2	struct pwm_config_t	728
34.4.3	struct pwm_fault_param_t	729
34.4.4	struct pwm_input_capture_param_t	729
<b>34.5</b>	<b>Enumeration Type Documentation</b>	<b>730</b>
34.5.1	pwm_submodule_t	730
34.5.2	pwm_value_register_t	730
34.5.3	pwm_clock_source_t	730
34.5.4	pwm_clock_prescale_t	730
34.5.5	pwm_force_output_trigger_t	731
34.5.6	pwm_init_source_t	731
34.5.7	pwm_load_frequency_t	731
34.5.8	pwm_fault_input_t	732
34.5.9	pwm_input_capture_edge_t	732
34.5.10	pwm_force_signal_t	732
34.5.11	pwm_chnl_pair_operation_t	732
34.5.12	pwm_register_reload_t	733
34.5.13	pwm_fault_recovery_mode_t	733
34.5.14	pwm_interrupt_enable_t	733
34.5.15	pwm_status_flags_t	734
34.5.16	pwm_mode_t	734
34.5.17	pwm_level_select_t	734
34.5.18	pwm_reload_source_select_t	735
34.5.19	pwm_fault_clear_t	735
34.5.20	pwm_module_control_t	735

# Contents

Section Number	Title	Page Number
<b>34.6</b>	<b>Function Documentation</b>	<b>735</b>
34.6.1	PWM_Init	735
34.6.2	PWM_Deinit	736
34.6.3	PWM_GetDefaultConfig	736
34.6.4	PWM_SetupPwm	736
34.6.5	PWM_UpdatePwmDutycycle	737
34.6.6	PWM_SetupInputCapture	737
34.6.7	PWM_SetupFaults	738
34.6.8	PWM_SetupForceSignal	738
34.6.9	PWM_EnableInterrupts	738
34.6.10	PWM_DisableInterrupts	739
34.6.11	PWM_GetEnabledInterrupts	739
34.6.12	PWM_GetStatusFlags	739
34.6.13	PWM_ClearStatusFlags	740
34.6.14	PWM_StartTimer	740
34.6.15	PWM_StopTimer	740
34.6.16	PWM_OutputTriggerEnable	741
34.6.17	PWM_SetupSwCtrlOut	741
34.6.18	PWM_SetPwmLdok	742
<b>Chapter</b>	<b>PXP: Pixel Pipeline</b>	
<b>35.1</b>	<b>Overview</b>	<b>743</b>
<b>35.2</b>	<b>Typical use case</b>	<b>743</b>
35.2.1	PXP normal operation	743
35.2.2	PXP operation queue	743
<b>35.3</b>	<b>Data Structure Documentation</b>	<b>749</b>
35.3.1	struct pxp_output_buffer_config_t	749
35.3.2	struct pxp_ps_buffer_config_t	750
35.3.3	struct pxp_as_buffer_config_t	751
35.3.4	struct pxp_as_blend_config_t	751
35.3.5	struct pxp_csc2_config_t	752
35.3.6	struct pxp_dither_final_lut_data_t	753
35.3.7	struct pxp_dither_config_t	754
<b>35.4</b>	<b>Enumeration Type Documentation</b>	<b>755</b>
35.4.1	_pxp_interrupt_enable	755
35.4.2	_pxp_flags	755
35.4.3	pxp_flip_mode_t	756
35.4.4	pxp_rotate_position_t	756
35.4.5	pxp_rotate_degree_t	756
35.4.6	pxp_interlaced_output_mode_t	756
35.4.7	pxp_output_pixel_format_t	756

# Contents

Section Number	Title	Page Number
35.4.8	pxp_ps_pixel_format_t . . . . .	757
35.4.9	pxp_as_pixel_format_t . . . . .	757
35.4.10	pxp_alpha_mode_t . . . . .	758
35.4.11	pxp_rop_mode_t . . . . .	758
35.4.12	pxp_block_size_t . . . . .	759
35.4.13	pxp_csc1_mode_t . . . . .	759
35.4.14	pxp_csc2_mode_t . . . . .	759
35.4.15	pxp_ram_t . . . . .	759
35.4.16	_pxp_dither_mode . . . . .	759
35.4.17	_pxp_dither_lut_mode . . . . .	760
35.4.18	_pxp_dither_matrix_size . . . . .	760
<b>35.5</b>	<b>Function Documentation . . . . .</b>	<b>760</b>
35.5.1	PXP_Init . . . . .	760
35.5.2	PXP_Deinit . . . . .	760
35.5.3	PXP_Reset . . . . .	760
35.5.4	PXP_Start . . . . .	761
35.5.5	PXP_EnableLcdHandShake . . . . .	761
35.5.6	PXP_EnableContinuousRun . . . . .	761
35.5.7	PXP_SetProcessBlockSize . . . . .	761
35.5.8	PXP_GetStatusFlags . . . . .	763
35.5.9	PXP_ClearStatusFlags . . . . .	763
35.5.10	PXP_GetAxiErrorId . . . . .	763
35.5.11	PXP_EnableInterrupts . . . . .	764
35.5.12	PXP_DisableInterrupts . . . . .	764
35.5.13	PXP_SetAlphaSurfaceBufferConfig . . . . .	765
35.5.14	PXP_SetAlphaSurfaceBlendConfig . . . . .	765
35.5.15	PXP_SetAlphaSurfaceOverlayColorKey . . . . .	765
35.5.16	PXP_EnableAlphaSurfaceOverlayColorKey . . . . .	766
35.5.17	PXP_SetAlphaSurfacePosition . . . . .	767
35.5.18	PXP_SetProcessSurfaceBackgroundColor . . . . .	767
35.5.19	PXP_SetProcessSurfaceBufferConfig . . . . .	767
35.5.20	PXP_SetProcessSurfaceScaler . . . . .	768
35.5.21	PXP_SetProcessSurfacePosition . . . . .	769
35.5.22	PXP_SetProcessSurfaceColorKey . . . . .	769
35.5.23	PXP_SetOutputBufferConfig . . . . .	769
35.5.24	PXP_SetOverwrittenAlphaValue . . . . .	770
35.5.25	PXP_EnableOverWrittenAlpha . . . . .	770
35.5.26	PXP_SetRotateConfig . . . . .	770
35.5.27	PXP_SetNextCommand . . . . .	771
35.5.28	PXP_IsNextCommandPending . . . . .	772
35.5.29	PXP_CancelNextCommand . . . . .	772
35.5.30	PXP_SetCsc1Mode . . . . .	772
35.5.31	PXP_EnableCsc1 . . . . .	773



# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>RTWDOG: 32-bit Watchdog Timer</b>	
<b>36.1</b>	<b>Overview</b> . . . . .	<b>775</b>
<b>36.2</b>	<b>Typical use case</b> . . . . .	<b>775</b>
<b>36.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>777</b>
36.3.1	struct rtwdog_work_mode_t . . . . .	777
36.3.2	struct rtwdog_config_t . . . . .	777
<b>36.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>777</b>
36.4.1	FSL_RTWDOG_DRIVER_VERSION . . . . .	777
<b>36.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>778</b>
36.5.1	rtwdog_clock_source_t . . . . .	778
36.5.2	rtwdog_clock_prescaler_t . . . . .	778
36.5.3	rtwdog_test_mode_t . . . . .	778
36.5.4	_rtwdog_interrupt_enable_t . . . . .	778
36.5.5	_rtwdog_status_flags_t . . . . .	778
<b>36.6</b>	<b>Function Documentation</b> . . . . .	<b>779</b>
36.6.1	RTWDOG_GetDefaultConfig . . . . .	779
36.6.2	RTWDOG_Init . . . . .	779
36.6.3	RTWDOG_Deinit . . . . .	780
36.6.4	RTWDOG_Enable . . . . .	780
36.6.5	RTWDOG_Disable . . . . .	780
36.6.6	RTWDOG_EnableInterrupts . . . . .	780
36.6.7	RTWDOG_DisableInterrupts . . . . .	782
36.6.8	RTWDOG_GetStatusFlags . . . . .	782
36.6.9	RTWDOG_ClearStatusFlags . . . . .	783
36.6.10	RTWDOG_SetTimeoutValue . . . . .	783
36.6.11	RTWDOG_SetWindowValue . . . . .	784
36.6.12	RTWDOG_Unlock . . . . .	784
36.6.13	RTWDOG_Refresh . . . . .	784
36.6.14	RTWDOG_GetCounterValue . . . . .	785
<b>Chapter</b>	<b>QTMR: Quad Timer Driver</b>	
<b>37.1</b>	<b>Overview</b> . . . . .	<b>787</b>
<b>37.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>790</b>
37.2.1	struct qtmr_config_t . . . . .	790
<b>37.3</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>790</b>
37.3.1	qtmr_primary_count_source_t . . . . .	790
37.3.2	qtmr_input_source_t . . . . .	791

# Contents

Section Number	Title	Page Number
37.3.3	qtmr_counting_mode_t . . . . .	791
37.3.4	qtmr_output_mode_t . . . . .	791
37.3.5	qtmr_input_capture_edge_t . . . . .	792
37.3.6	qtmr_preload_control_t . . . . .	792
37.3.7	qtmr_debug_action_t . . . . .	792
37.3.8	qtmr_interrupt_enable_t . . . . .	792
37.3.9	qtmr_status_flags_t . . . . .	793
<b>37.4</b>	<b>Function Documentation . . . . .</b>	<b>793</b>
37.4.1	QTMR_Init . . . . .	793
37.4.2	QTMR_Deinit . . . . .	793
37.4.3	QTMR_GetDefaultConfig . . . . .	793
37.4.4	QTMR_SetupPwm . . . . .	794
37.4.5	QTMR_SetupInputCapture . . . . .	794
37.4.6	QTMR_EnableInterrupts . . . . .	795
37.4.7	QTMR_DisableInterrupts . . . . .	795
37.4.8	QTMR_GetEnabledInterrupts . . . . .	795
37.4.9	QTMR_GetStatus . . . . .	795
37.4.10	QTMR_ClearStatusFlags . . . . .	796
37.4.11	QTMR_SetTimerPeriod . . . . .	796
37.4.12	QTMR_GetCurrentTimerCount . . . . .	796
37.4.13	QTMR_StartTimer . . . . .	797
37.4.14	QTMR_StopTimer . . . . .	797
<b>Chapter</b>	<b>SAI: Serial Audio Interface</b>	
<b>38.1</b>	<b>Overview . . . . .</b>	<b>799</b>
<b>38.2</b>	<b>Typical use case . . . . .</b>	<b>799</b>
38.2.1	SAI Send/receive using an interrupt method . . . . .	799
38.2.2	SAI Send/receive using a DMA method . . . . .	799
<b>38.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>805</b>
38.3.1	struct sai_config_t . . . . .	805
38.3.2	struct sai_transfer_format_t . . . . .	805
38.3.3	struct sai_transfer_t . . . . .	806
38.3.4	struct _sai_handle . . . . .	806
<b>38.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>807</b>
38.4.1	SAI_XFER_QUEUE_SIZE . . . . .	807
<b>38.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>807</b>
38.5.1	_sai_status_t . . . . .	807
38.5.2	sai_protocol_t . . . . .	807
38.5.3	sai_master_slave_t . . . . .	807

# Contents

Section Number	Title	Page Number
38.5.4	sai_mono_stereo_t . . . . .	808
38.5.5	sai_data_order_t . . . . .	808
38.5.6	sai_clock_polarity_t . . . . .	808
38.5.7	sai_sync_mode_t . . . . .	808
38.5.8	sai_mclk_source_t . . . . .	808
38.5.9	sai_bclk_source_t . . . . .	809
38.5.10	_sai_interrupt_enable_t . . . . .	809
38.5.11	_sai_dma_enable_t . . . . .	809
38.5.12	_sai_flags . . . . .	809
38.5.13	sai_reset_type_t . . . . .	809
38.5.14	sai_fifo_packing_t . . . . .	810
38.5.15	sai_sample_rate_t . . . . .	810
38.5.16	sai_word_width_t . . . . .	810
<b>38.6</b>	<b>Function Documentation . . . . .</b>	<b>810</b>
38.6.1	SAI_TxInit . . . . .	810
38.6.2	SAI_RxInit . . . . .	811
38.6.3	SAI_TxGetDefaultConfig . . . . .	811
38.6.4	SAI_RxGetDefaultConfig . . . . .	812
38.6.5	SAI_Deinit . . . . .	812
38.6.6	SAI_TxReset . . . . .	812
38.6.7	SAI_RxReset . . . . .	812
38.6.8	SAI_TxEnable . . . . .	813
38.6.9	SAI_RxEnable . . . . .	813
38.6.10	SAI_TxGetStatusFlag . . . . .	813
38.6.11	SAI_TxClearStatusFlags . . . . .	813
38.6.12	SAI_RxGetStatusFlag . . . . .	814
38.6.13	SAI_RxClearStatusFlags . . . . .	814
38.6.14	SAI_TxSoftwareReset . . . . .	814
38.6.15	SAI_RxSoftwareReset . . . . .	815
38.6.16	SAI_TxSetChannelFIFOMask . . . . .	815
38.6.17	SAI_RxSetChannelFIFOMask . . . . .	815
38.6.18	SAI_TxSetDataOrder . . . . .	815
38.6.19	SAI_RxSetDataOrder . . . . .	816
38.6.20	SAI_TxSetBitClockPolarity . . . . .	816
38.6.21	SAI_RxSetBitClockPolarity . . . . .	816
38.6.22	SAI_TxSetFrameSyncPolarity . . . . .	816
38.6.23	SAI_RxSetFrameSyncPolarity . . . . .	817
38.6.24	SAI_TxSetFIFOPacking . . . . .	817
38.6.25	SAI_RxSetFIFOPacking . . . . .	817
38.6.26	SAI_TxSetFIFOErrorContinue . . . . .	817
38.6.27	SAI_RxSetFIFOErrorContinue . . . . .	818
38.6.28	SAI_TxEnableInterrupts . . . . .	818
38.6.29	SAI_RxEnableInterrupts . . . . .	818
38.6.30	SAI_TxDisableInterrupts . . . . .	819

# Contents

Section Number	Title	Page Number
38.6.31	SAI_RxDisableInterrupts . . . . .	820
38.6.32	SAI_TxEnableDMA . . . . .	820
38.6.33	SAI_RxEnableDMA . . . . .	820
38.6.34	SAI_TxGetDataRegisterAddress . . . . .	821
38.6.35	SAI_RxGetDataRegisterAddress . . . . .	822
38.6.36	SAI_TxSetFormat . . . . .	822
38.6.37	SAI_RxSetFormat . . . . .	823
38.6.38	SAI_WriteBlocking . . . . .	823
38.6.39	SAI_WriteData . . . . .	823
38.6.40	SAI_ReadBlocking . . . . .	824
38.6.41	SAI_ReadData . . . . .	824
38.6.42	SAI_TransferTxCreateHandle . . . . .	824
38.6.43	SAI_TransferRxCreateHandle . . . . .	825
38.6.44	SAI_TransferTxSetFormat . . . . .	825
38.6.45	SAI_TransferRxSetFormat . . . . .	826
38.6.46	SAI_TransferSendNonBlocking . . . . .	826
38.6.47	SAI_TransferReceiveNonBlocking . . . . .	827
38.6.48	SAI_TransferGetSendCount . . . . .	827
38.6.49	SAI_TransferGetReceiveCount . . . . .	828
38.6.50	SAI_TransferAbortSend . . . . .	828
38.6.51	SAI_TransferAbortReceive . . . . .	829
38.6.52	SAI_TransferTerminateSend . . . . .	829
38.6.53	SAI_TransferTerminateReceive . . . . .	829
38.6.54	SAI_TransferTxHandleIRQ . . . . .	830
38.6.55	SAI_TransferRxHandleIRQ . . . . .	830
<b>38.7</b>	<b>SAI DMA Driver . . . . .</b>	<b>831</b>
38.7.1	Overview . . . . .	831
38.7.2	Data Structure Documentation . . . . .	832
38.7.3	Function Documentation . . . . .	832
<b>38.8</b>	<b>SAI eDMA Driver . . . . .</b>	<b>838</b>
38.8.1	Overview . . . . .	838
38.8.2	Data Structure Documentation . . . . .	839
38.8.3	Function Documentation . . . . .	840
<b>38.9</b>	<b>SAI SDMA Driver . . . . .</b>	<b>847</b>
38.9.1	Overview . . . . .	847
38.9.2	Data Structure Documentation . . . . .	848
38.9.3	Function Documentation . . . . .	849
<b>Chapter</b>	<b>SEMC: Smart External DRAM Controller Driver</b>	
<b>39.1</b>	<b>Overview . . . . .</b>	<b>855</b>

# Contents

Section Number	Title	Page Number
<b>39.2</b>	<b>Typical use case</b> . . . . .	<b>855</b>
<b>39.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>861</b>
39.3.1	struct semc_sdram_config_t . . . . .	861
39.3.2	struct semc_nand_timing_config_t . . . . .	863
39.3.3	struct semc_nand_config_t . . . . .	865
39.3.4	struct semc_nor_config_t . . . . .	866
39.3.5	struct semc_sram_config_t . . . . .	869
39.3.6	struct semc_dbi_config_t . . . . .	872
39.3.7	struct semc_queuea_weight_t . . . . .	874
39.3.8	struct semc_queueb_weight_t . . . . .	875
39.3.9	struct semc_axi_queueweight_t . . . . .	875
39.3.10	struct semc_config_t . . . . .	876
<b>39.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>876</b>
39.4.1	FSL_SEMC_DRIVER_VERSION . . . . .	876
<b>39.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>876</b>
39.5.1	_semc_status . . . . .	876
39.5.2	semc_mem_type_t . . . . .	876
39.5.3	semc_waitready_polarity_t . . . . .	877
39.5.4	semc_sdram_cs_t . . . . .	877
39.5.5	semc_nand_access_type_t . . . . .	877
39.5.6	semc_interrupt_enable_t . . . . .	877
39.5.7	semc_ipcmd_datasize_t . . . . .	877
39.5.8	semc_refresh_time_t . . . . .	878
39.5.9	semc_caslatency_t . . . . .	878
39.5.10	semc_sdram_column_bit_num_t . . . . .	878
39.5.11	sem_sdram_burst_len_t . . . . .	878
39.5.12	semc_nand_column_bit_num_t . . . . .	878
39.5.13	sem_nand_burst_len_t . . . . .	879
39.5.14	semc_norsram_column_bit_num_t . . . . .	879
39.5.15	sem_norsram_burst_len_t . . . . .	879
39.5.16	semc_dbi_column_bit_num_t . . . . .	880
39.5.17	sem_dbi_burst_len_t . . . . .	880
39.5.18	semc_iomux_pin . . . . .	880
39.5.19	semc_iomux_nora27_pin . . . . .	881
39.5.20	smec_port_size_t . . . . .	881
39.5.21	semc_addr_mode_t . . . . .	881
39.5.22	semc_dqs_mode_t . . . . .	881
39.5.23	semc_adv_polarity_t . . . . .	881
39.5.24	semc_rdy_polarity_t . . . . .	882
39.5.25	semc_ipcmd_nand_addrmode_t . . . . .	882
39.5.26	semc_ipcmd_nand_cmdmode_t . . . . .	882
39.5.27	semc_nand_address_option_t . . . . .	882

# Contents

Section Number	Title	Page Number
39.5.28	semc_ipcmd_nor_dbi_t . . . . .	883
39.5.29	semc_ipcmd_sram_t . . . . .	883
39.5.30	semc_ipcmd_sdram_t . . . . .	883
<b>39.6</b>	<b>Function Documentation . . . . .</b>	<b>883</b>
39.6.1	SEMC_GetDefaultConfig . . . . .	883
39.6.2	SEMC_Init . . . . .	884
39.6.3	SEMC_Deinit . . . . .	884
39.6.4	SEMC_ConfigureSDRAM . . . . .	884
39.6.5	SEMC_ConfigureNAND . . . . .	884
39.6.6	SEMC_ConfigureNOR . . . . .	885
39.6.7	SEMC_ConfigureSRAM . . . . .	885
39.6.8	SEMC_ConfigureDBI . . . . .	885
39.6.9	SEMC_EnableInterrupts . . . . .	886
39.6.10	SEMC_DisableInterrupts . . . . .	886
39.6.11	SEMC_GetStatusFlag . . . . .	886
39.6.12	SEMC_ClearStatusFlags . . . . .	887
39.6.13	SEMC_IsInIdle . . . . .	887
39.6.14	SEMC_SendIPCommand . . . . .	888
39.6.15	SEMC_BuildNandIPCommand . . . . .	888
39.6.16	SEMC_IsNandReady . . . . .	888
39.6.17	SEMC_IPCommandNandWrite . . . . .	889
39.6.18	SEMC_IPCommandNandRead . . . . .	889
39.6.19	SEMC_IPCommandNorWrite . . . . .	889
39.6.20	SEMC_IPCommandNorRead . . . . .	890
<b>Chapter</b>	<b>SNVS: Secure Non-Volatile Storage</b>	
<b>40.1</b>	<b>Overview . . . . .</b>	<b>891</b>
<b>40.2</b>	<b>Secure Non-Volatile Storage High-Power . . . . .</b>	<b>892</b>
40.2.1	Overview . . . . .	892
40.2.2	SNVS_HP Driver Initialization and Configuration . . . . .	892
40.2.3	SNVS_HP Driver Examples . . . . .	892
40.2.4	Data Structure Documentation . . . . .	894
40.2.5	Enumeration Type Documentation . . . . .	895
40.2.6	Function Documentation . . . . .	895
40.2.7	Variable Documentation . . . . .	899
<b>Chapter</b>	<b>SPDIF: Sony/Philips Digital Interface</b>	
<b>41.1</b>	<b>Overview . . . . .</b>	<b>901</b>
<b>41.2</b>	<b>Typical use case . . . . .</b>	<b>901</b>
41.2.1	SPDIF Send/receive using an interrupt method . . . . .	901

# Contents

Section Number	Title	Page Number
41.2.2	SPDIF Send/receive using a DMA method . . . . .	901
<b>41.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>906</b>
41.3.1	struct spdif_config_t . . . . .	906
41.3.2	struct spdif_transfer_t . . . . .	906
41.3.3	struct _spdif_handle . . . . .	907
<b>41.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>907</b>
41.4.1	SPDIF_XFER_QUEUE_SIZE . . . . .	907
<b>41.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>907</b>
41.5.1	_spdif_status_t . . . . .	907
41.5.2	spdif_rxfull_select_t . . . . .	908
41.5.3	spdif_txempty_select_t . . . . .	908
41.5.4	spdif_uchannel_source_t . . . . .	908
41.5.5	spdif_gain_select_t . . . . .	908
41.5.6	spdif_tx_source_t . . . . .	909
41.5.7	spdif_validity_config_t . . . . .	909
41.5.8	_spdif_interrupt_enable_t . . . . .	909
41.5.9	_spdif_dma_enable_t . . . . .	910
<b>41.6</b>	<b>Function Documentation . . . . .</b>	<b>910</b>
41.6.1	SPDIF_Init . . . . .	910
41.6.2	SPDIF_GetDefaultConfig . . . . .	910
41.6.3	SPDIF_Deinit . . . . .	911
41.6.4	SPDIF_TxFIFOReset . . . . .	911
41.6.5	SPDIF_RxFIFOReset . . . . .	911
41.6.6	SPDIF_TxEnable . . . . .	911
41.6.7	SPDIF_RxEnable . . . . .	912
41.6.8	SPDIF_GetStatusFlag . . . . .	912
41.6.9	SPDIF_ClearStatusFlags . . . . .	912
41.6.10	SPDIF_EnableInterrupts . . . . .	913
41.6.11	SPDIF_DisableInterrupts . . . . .	914
41.6.12	SPDIF_EnableDMA . . . . .	914
41.6.13	SPDIF_TxGetLeftDataRegisterAddress . . . . .	915
41.6.14	SPDIF_TxGetRightDataRegisterAddress . . . . .	916
41.6.15	SPDIF_RxGetLeftDataRegisterAddress . . . . .	916
41.6.16	SPDIF_RxGetRightDataRegisterAddress . . . . .	916
41.6.17	SPDIF_TxSetSampleRate . . . . .	917
41.6.18	SPDIF_GetRxSampleRate . . . . .	917
41.6.19	SPDIF_WriteBlocking . . . . .	917
41.6.20	SPDIF_WriteLeftData . . . . .	918
41.6.21	SPDIF_WriteRightData . . . . .	918
41.6.22	SPDIF_WriteChannelStatusHigh . . . . .	918
41.6.23	SPDIF_WriteChannelStatusLow . . . . .	918

# Contents

Section Number	Title	Page Number
41.6.24	SPDIF_ReadBlocking . . . . .	919
41.6.25	SPDIF_ReadLeftData . . . . .	919
41.6.26	SPDIF_ReadRightData . . . . .	919
41.6.27	SPDIF_ReadChannelStatusHigh . . . . .	920
41.6.28	SPDIF_ReadChannelStatusLow . . . . .	920
41.6.29	SPDIF_ReadQChannel . . . . .	920
41.6.30	SPDIF_ReadUChannel . . . . .	921
41.6.31	SPDIF_TransferTxCreateHandle . . . . .	921
41.6.32	SPDIF_TransferRxCreateHandle . . . . .	921
41.6.33	SPDIF_TransferSendNonBlocking . . . . .	922
41.6.34	SPDIF_TransferReceiveNonBlocking . . . . .	922
41.6.35	SPDIF_TransferGetSendCount . . . . .	923
41.6.36	SPDIF_TransferGetReceiveCount . . . . .	923
41.6.37	SPDIF_TransferAbortSend . . . . .	924
41.6.38	SPDIF_TransferAbortReceive . . . . .	924
41.6.39	SPDIF_TransferTxHandleIRQ . . . . .	924
41.6.40	SPDIF_TransferRxHandleIRQ . . . . .	925
<b>41.7</b>	<b>SPDIF eDMA Driver . . . . .</b>	<b>926</b>
41.7.1	Overview . . . . .	926
41.7.2	Data Structure Documentation . . . . .	927
41.7.3	Function Documentation . . . . .	928
<b>Chapter</b>	<b>SRC: System Reset Controller Driver</b>	
<b>42.1</b>	<b>Overview . . . . .</b>	<b>933</b>
<b>42.2</b>	<b>Macro Definition Documentation . . . . .</b>	<b>934</b>
42.2.1	FSL_SRC_DRIVER_VERSION . . . . .	934
<b>42.3</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>934</b>
42.3.1	_src_reset_status_flags . . . . .	934
42.3.2	src_warm_reset_bypass_count_t . . . . .	935
<b>42.4</b>	<b>Function Documentation . . . . .</b>	<b>935</b>
42.4.1	SRC_EnableWDOG3Reset . . . . .	935
42.4.2	SRC_EnableCoreDebugResetAfterPowerGate . . . . .	935
42.4.3	SRC_DoSoftwareResetARMCore0 . . . . .	935
42.4.4	SRC_GetSoftwareResetARMCore0Done . . . . .	936
42.4.5	SRC_EnableWDOGReset . . . . .	936
42.4.6	SRC_EnableLockupReset . . . . .	936
42.4.7	SRC_GetBootModeWord1 . . . . .	937
42.4.8	SRC_GetBootModeWord2 . . . . .	937
42.4.9	SRC_GetResetStatusFlags . . . . .	937
42.4.10	SRC_ClearResetStatusFlags . . . . .	938



# Contents

Section Number	Title	Page Number
42.4.11	SRC_SetGeneralPurposeRegister . . . . .	938
42.4.12	SRC_GetGeneralPurposeRegister . . . . .	938
<b>Chapter</b>	<b>TRNG: True Random Number Generator</b>	
<b>43.1</b>	<b>Overview . . . . .</b>	<b>941</b>
<b>43.2</b>	<b>TRNG Initialization . . . . .</b>	<b>941</b>
<b>43.3</b>	<b>Get random data from TRNG . . . . .</b>	<b>941</b>
<b>43.4</b>	<b>Data Structure Documentation . . . . .</b>	<b>942</b>
43.4.1	struct trng_statistical_check_limit_t . . . . .	942
43.4.2	struct trng_config_t . . . . .	943
<b>43.5</b>	<b>Macro Definition Documentation . . . . .</b>	<b>945</b>
43.5.1	FSL_TRNG_DRIVER_VERSION . . . . .	945
<b>43.6</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>945</b>
43.6.1	trng_sample_mode_t . . . . .	945
43.6.2	trng_clock_mode_t . . . . .	946
43.6.3	trng_ring_osc_div_t . . . . .	946
<b>43.7</b>	<b>Function Documentation . . . . .</b>	<b>946</b>
43.7.1	TRNG_GetDefaultConfig . . . . .	946
43.7.2	TRNG_Init . . . . .	948
43.7.3	TRNG_Deinit . . . . .	948
43.7.4	TRNG_GetRandomData . . . . .	948
<b>Chapter</b>	<b>TSC: Touch Screen Controller Driver</b>	
<b>44.1</b>	<b>Overview . . . . .</b>	<b>951</b>
<b>44.2</b>	<b>Typical use case . . . . .</b>	<b>951</b>
44.2.1	4-wire Polling Configuration . . . . .	951
44.2.2	4-wire Interrupt Configuration . . . . .	951
<b>44.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>953</b>
44.3.1	struct tsc_config_t . . . . .	953
<b>44.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>954</b>
44.4.1	FSL_TSC_DRIVER_VERSION . . . . .	954
<b>44.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>954</b>
44.5.1	tsc_detection_mode_t . . . . .	954
44.5.2	tsc_coordinate_value_selection_t . . . . .	954

# Contents

Section Number	Title	Page Number
44.5.3	<code>_tsc_interrupt_signal_mask</code> . . . . .	954
44.5.4	<code>_tsc_interrupt_mask</code> . . . . .	955
44.5.5	<code>_tsc_interrupt_status_flag_mask</code> . . . . .	955
44.5.6	<code>_tsc_adc_status_flag_mask</code> . . . . .	955
44.5.7	<code>_tsc_status_flag_mask</code> . . . . .	955
44.5.8	<code>tsc_state_machine_t</code> . . . . .	956
44.5.9	<code>tsc_glitch_threshold_t</code> . . . . .	956
44.5.10	<code>tsc_trigger_signal_t</code> . . . . .	956
44.5.11	<code>tsc_port_source_t</code> . . . . .	957
44.5.12	<code>tsc_port_mode_t</code> . . . . .	957
<b>44.6</b>	<b>Function Documentation</b> . . . . .	<b>957</b>
44.6.1	<code>TSC_Init</code> . . . . .	957
44.6.2	<code>TSC_Deinit</code> . . . . .	957
<b>44.7</b>	<b>Variable Documentation</b> . . . . .	<b>958</b>
44.7.1	<code>enableAutoMeasure</code> . . . . .	958
44.7.2	<code>measureDelayTime</code> . . . . .	958
44.7.3	<code>prechargeTime</code> . . . . .	958
44.7.4	<code>detectionMode</code> . . . . .	958
<b>Chapter</b>	<b>USDHC: ultra Secured Digital Host Controller Driver</b>	
<b>45.1</b>	<b>Overview</b> . . . . .	<b>959</b>
<b>45.2</b>	<b>Typical use case</b> . . . . .	<b>959</b>
45.2.1	USDHC Operation . . . . .	959
<b>45.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>969</b>
45.3.1	<code>struct usdhc_adma2_descriptor_t</code> . . . . .	969
45.3.2	<code>struct usdhc_capability_t</code> . . . . .	970
45.3.3	<code>struct usdhc_boot_config_t</code> . . . . .	970
45.3.4	<code>struct usdhc_config_t</code> . . . . .	971
45.3.5	<code>struct usdhc_data_t</code> . . . . .	971
45.3.6	<code>struct usdhc_command_t</code> . . . . .	972
45.3.7	<code>struct usdhc_adma_config_t</code> . . . . .	972
45.3.8	<code>struct usdhc_transfer_t</code> . . . . .	973
45.3.9	<code>struct usdhc_transfer_callback_t</code> . . . . .	973
45.3.10	<code>struct _usdhc_handle</code> . . . . .	973
45.3.11	<code>struct usdhc_host_t</code> . . . . .	974
<b>45.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>974</b>
45.4.1	<code>FSL_USDHC_DRIVER_VERSION</code> . . . . .	974
<b>45.5</b>	<b>Typedef Documentation</b> . . . . .	<b>974</b>
45.5.1	<code>usdhc_adma1_descriptor_t</code> . . . . .	974

# Contents

Section Number	Title	Page Number
45.5.2	usdhc_transfer_function_t . . . . .	974
<b>45.6</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>974</b>
45.6.1	_usdhc_status . . . . .	974
45.6.2	_usdhc_capability_flag . . . . .	975
45.6.3	_usdhc_wakeup_event . . . . .	975
45.6.4	_usdhc_reset . . . . .	975
45.6.5	_usdhc_transfer_flag . . . . .	976
45.6.6	_usdhc_present_status_flag . . . . .	976
45.6.7	_usdhc_interrupt_status_flag . . . . .	977
45.6.8	_usdhc_auto_command12_error_status_flag . . . . .	977
45.6.9	_usdhc_standard_tuning . . . . .	978
45.6.10	_usdhc_adma_error_status_flag . . . . .	978
45.6.11	_usdhc_adma_error_state . . . . .	978
45.6.12	_usdhc_force_event . . . . .	978
45.6.13	usdhc_data_bus_width_t . . . . .	979
45.6.14	usdhc_endian_mode_t . . . . .	979
45.6.15	usdhc_dma_mode_t . . . . .	979
45.6.16	_usdhc_sdio_control_flag . . . . .	980
45.6.17	usdhc_boot_mode_t . . . . .	980
45.6.18	usdhc_card_command_type_t . . . . .	980
45.6.19	usdhc_card_response_type_t . . . . .	980
45.6.20	_usdhc_adma1_descriptor_flag . . . . .	981
45.6.21	_usdhc_adma2_descriptor_flag . . . . .	981
45.6.22	_usdhc_adma_flag . . . . .	981
45.6.23	usdhc_burst_len_t . . . . .	982
45.6.24	_usdhc_transfer_data_type . . . . .	982
<b>45.7</b>	<b>Function Documentation . . . . .</b>	<b>982</b>
45.7.1	USDHC_Init . . . . .	982
45.7.2	USDHC_Deinit . . . . .	982
45.7.3	USDHC_Reset . . . . .	983
45.7.4	USDHC_SetAdmaTableConfig . . . . .	983
45.7.5	USDHC_SetInternalDmaConfig . . . . .	984
45.7.6	USDHC_SetADMA2Descriptor . . . . .	985
45.7.7	USDHC_SetADMA1Descriptor . . . . .	985
45.7.8	USDHC_EnableInternalDMA . . . . .	986
45.7.9	USDHC_EnableInterruptStatus . . . . .	986
45.7.10	USDHC_DisableInterruptStatus . . . . .	986
45.7.11	USDHC_EnableInterruptSignal . . . . .	987
45.7.12	USDHC_DisableInterruptSignal . . . . .	987
45.7.13	USDHC_GetInterruptStatusFlags . . . . .	987
45.7.14	USDHC_ClearInterruptStatusFlags . . . . .	987
45.7.15	USDHC_GetAutoCommand12ErrorStatusFlags . . . . .	988
45.7.16	USDHC_GetAdmaErrorStatusFlags . . . . .	988

# Contents

Section Number	Title	Page Number
45.7.17	USDHC_GetPresentStatusFlags . . . . .	988
45.7.18	USDHC_GetCapability . . . . .	989
45.7.19	USDHC_ForceClockOn . . . . .	990
45.7.20	USDHC_SetSdClock . . . . .	990
45.7.21	USDHC_SetCardActive . . . . .	990
45.7.22	USDHC_AssertHardwareReset . . . . .	991
45.7.23	USDHC_SetDataBusWidth . . . . .	991
45.7.24	USDHC_WriteData . . . . .	991
45.7.25	USDHC_ReadData . . . . .	991
45.7.26	USDHC_SendCommand . . . . .	992
45.7.27	USDHC_EnableWakeupEvent . . . . .	992
45.7.28	USDHC_CardDetectByData3 . . . . .	992
45.7.29	USDHC_DetectCardInsert . . . . .	992
45.7.30	USDHC_EnableSdioControl . . . . .	993
45.7.31	USDHC_SetContinueRequest . . . . .	993
45.7.32	USDHC_RequestStopAtBlockGap . . . . .	993
45.7.33	USDHC_SetMmcBootConfig . . . . .	993
45.7.34	USDHC_EnableMmcBoot . . . . .	994
45.7.35	USDHC_SetForceEvent . . . . .	994
45.7.36	UDSHC_SelectVoltage . . . . .	994
45.7.37	USDHC_RequestTuningForSDR50 . . . . .	994
45.7.38	USDHC_RequestReTuning . . . . .	995
45.7.39	USDHC_EnableAutoTuning . . . . .	995
45.7.40	USDHC_SetRetuningTimer . . . . .	995
45.7.41	USDHC_EnableAutoTuningForCmdAndData . . . . .	995
45.7.42	USDHC_EnableManualTuning . . . . .	996
45.7.43	USDHC_AdjustDelayForManualTuning . . . . .	996
45.7.44	USDHC_EnableStandardTuning . . . . .	996
45.7.45	USDHC_GetExecuteStdTuningStatus . . . . .	997
45.7.46	USDHC_CheckStdTuningResult . . . . .	997
45.7.47	USDHC_CheckTuningError . . . . .	997
45.7.48	USDHC_EnableDDRMMode . . . . .	997
45.7.49	USDHC_TransferBlocking . . . . .	998
45.7.50	USDHC_TransferCreateHandle . . . . .	998
45.7.51	USDHC_TransferNonBlocking . . . . .	999
45.7.52	USDHC_TransferHandleIRQ . . . . .	1000

## Chapter **WDOG: Watchdog Timer Driver**

<b>46.1</b>	<b>Overview . . . . .</b>	<b>1001</b>
<b>46.2</b>	<b>Typical use case . . . . .</b>	<b>1001</b>
<b>46.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>1003</b>
46.3.1	struct wdog_work_mode_t . . . . .	1003

## Contents

Section Number	Title	Page Number
46.3.2	struct wdog_config_t . . . . .	.1003
46.3.3	struct wdog_test_config_t . . . . .	.1004
<b>46.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>.1004</b>
46.4.1	FSL_WDOG_DRIVER_VERSION . . . . .	.1004
<b>46.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>.1004</b>
46.5.1	wdog_clock_source_t . . . . .	.1004
46.5.2	wdog_clock_prescaler_t . . . . .	.1004
46.5.3	wdog_test_mode_t . . . . .	.1005
46.5.4	wdog_tested_byte_t . . . . .	.1005
46.5.5	_wdog_interrupt_enable_t . . . . .	.1005
46.5.6	_wdog_status_flags_t . . . . .	.1005
<b>46.6</b>	<b>Function Documentation . . . . .</b>	<b>.1005</b>
46.6.1	WDOG_GetDefaultConfig . . . . .	.1005
46.6.2	WDOG_Init . . . . .	.1006
46.6.3	WDOG_Deinit . . . . .	.1006
46.6.4	WDOG_SetTestModeConfig . . . . .	.1007
46.6.5	WDOG_Enable . . . . .	.1007
46.6.6	WDOG_Disable . . . . .	.1007
46.6.7	WDOG_EnableInterrupts . . . . .	.1008
46.6.8	WDOG_DisableInterrupts . . . . .	.1008
46.6.9	WDOG_GetStatusFlags . . . . .	.1008
46.6.10	WDOG_ClearStatusFlags . . . . .	.1009
46.6.11	WDOG_SetTimeoutValue . . . . .	.1009
46.6.12	WDOG_SetWindowValue . . . . .	.1010
46.6.13	WDOG_Unlock . . . . .	.1010
46.6.14	WDOG_Refresh . . . . .	.1010
46.6.15	WDOG_GetResetCount . . . . .	.1011
46.6.16	WDOG_ClearResetCount . . . . .	.1012
<b>Chapter</b>	<b>XBARA: Inter-Peripheral Crossbar Switch</b>	
<b>47.1</b>	<b>Overview . . . . .</b>	<b>.1013</b>
<b>47.2</b>	<b>Function . . . . .</b>	<b>.1013</b>
47.2.1	XBARA Initialization . . . . .	.1013
47.2.2	Call diagram . . . . .	.1013
<b>47.3</b>	<b>Typical use case . . . . .</b>	<b>.1013</b>
<b>47.4</b>	<b>Data Structure Documentation . . . . .</b>	<b>.1014</b>
47.4.1	struct xbara_control_config_t . . . . .	.1014
<b>47.5</b>	<b>Macro Definition Documentation . . . . .</b>	<b>.1015</b>

# Contents

Section Number	Title	Page Number
47.5.1	FSL_XBARA_DRIVER_VERSION . . . . .	.1015
<b>47.6</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>.1015</b>
47.6.1	xbara_active_edge_t . . . . .	.1015
47.6.2	xbara_request_t . . . . .	.1015
47.6.3	xbara_status_flag_t . . . . .	.1015
<b>47.7</b>	<b>Function Documentation . . . . .</b>	<b>.1016</b>
47.7.1	XBARA_Init . . . . .	.1016
47.7.2	XBARA_Deinit . . . . .	.1016
47.7.3	XBARA_SetSignalsConnection . . . . .	.1016
47.7.4	XBARA_GetStatusFlags . . . . .	.1017
47.7.5	XBARA_ClearStatusFlags . . . . .	.1017
47.7.6	XBARA_SetOutputSignalConfig . . . . .	.1017
<b>Chapter</b>	<b>XBARB: Inter-Peripheral Crossbar Switch</b>	
<b>48.1</b>	<b>Overview . . . . .</b>	<b>.1019</b>
<b>48.2</b>	<b>Function groups . . . . .</b>	<b>.1019</b>
48.2.1	XBARB Initialization . . . . .	.1019
48.2.2	Call diagram . . . . .	.1019
<b>48.3</b>	<b>Typical use case . . . . .</b>	<b>.1019</b>
<b>48.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>.1020</b>
48.4.1	FSL_XBARB_DRIVER_VERSION . . . . .	.1020
<b>48.5</b>	<b>Function Documentation . . . . .</b>	<b>.1020</b>
48.5.1	XBARB_Init . . . . .	.1020
48.5.2	XBARB_Deinit . . . . .	.1020
48.5.3	XBARB_SetSignalsConnection . . . . .	.1020
<b>Chapter</b>	<b>Clock Driver</b>	
<b>Chapter</b>	<b>DMA Manager</b>	
<b>50.1</b>	<b>Overview . . . . .</b>	<b>.1025</b>
<b>50.2</b>	<b>Function groups . . . . .</b>	<b>.1025</b>
50.2.1	DMAMGR Initialization and De-initialization . . . . .	.1025
50.2.2	DMAMGR Operation . . . . .	.1025
<b>50.3</b>	<b>Typical use case . . . . .</b>	<b>.1025</b>
50.3.1	DMAMGR static channel allocattion . . . . .	.1025
50.3.2	DMAMGR dynamic channel allocation . . . . .	.1025

# Contents

Section Number	Title	Page Number
<b>50.4</b>	<b>Data Structure Documentation</b>	<b>.1026</b>
50.4.1	struct dmamanager_handle_t	.1026
<b>50.5</b>	<b>Macro Definition Documentation</b>	<b>.1027</b>
50.5.1	DMAMGR_DYNAMIC_ALLOCATE	.1027
<b>50.6</b>	<b>Enumeration Type Documentation</b>	<b>.1027</b>
50.6.1	_dma_manager_status	.1027
<b>50.7</b>	<b>Function Documentation</b>	<b>.1027</b>
50.7.1	DMAMGR_Init	.1027
50.7.2	DMAMGR_Deinit	.1028
50.7.3	DMAMGR_RequestChannel	.1028
50.7.4	DMAMGR_ReleaseChannel	.1029
50.7.5	DMAMGR_IsChannelOccupied	.1030
<b>Chapter</b>	<b>Debug Console</b>	
<b>51.1</b>	<b>Overview</b>	<b>.1031</b>
<b>51.2</b>	<b>Function groups</b>	<b>.1031</b>
51.2.1	Initialization	.1031
51.2.2	Advanced Feature	.1032
<b>51.3</b>	<b>Typical use case</b>	<b>.1035</b>
<b>51.4</b>	<b>Data Structure Documentation</b>	<b>.1038</b>
51.4.1	struct io_state_t	.1038
<b>51.5</b>	<b>Macro Definition Documentation</b>	<b>.1038</b>
51.5.1	SDK_DEBUGCONSOLE	.1038
51.5.2	SDK_DEBUGCONSOLE_UART	.1038
<b>51.6</b>	<b>Typedef Documentation</b>	<b>.1038</b>
51.6.1	notify	.1038
<b>51.7</b>	<b>Enumeration Type Documentation</b>	<b>.1039</b>
51.7.1	_swo_protocol	.1039
<b>51.8</b>	<b>Function Documentation</b>	<b>.1039</b>
51.8.1	DbgConsole_Init	.1039
51.8.2	DbgConsole_Deinit	.1040
51.8.3	DbgConsole_Printf	.1040
51.8.4	DbgConsole_Putchar	.1040
51.8.5	DbgConsole_Scanf	.1041
51.8.6	DbgConsole_Getchar	.1041

# Contents

Section Number	Title	Page Number
51.8.7	DbgConsole_Flush . . . . .	.1041
51.8.8	IO_Init . . . . .	.1041
51.8.9	IO_Deinit . . . . .	.1042
51.8.10	IO_Transfer . . . . .	.1042
51.8.11	IO_WaitIdle . . . . .	.1042
51.8.12	SWO_Init . . . . .	.1043
51.8.13	SWO_Deinit . . . . .	.1044
51.8.14	SWO_SendBlocking . . . . .	.1044
51.8.15	LOG_Init . . . . .	.1044
51.8.16	LOG_Deinit . . . . .	.1045
51.8.17	LOG_Push . . . . .	.1045
51.8.18	LOG_ReadLine . . . . .	.1046
51.8.19	LOG_ReadCharacter . . . . .	.1046
51.8.20	LOG_WaitIdle . . . . .	.1046
51.8.21	LOG_Pop . . . . .	.1046
51.8.22	StrFormatPrintf . . . . .	.1047
51.8.23	StrFormatScanf . . . . .	.1047
<b>51.9</b>	<b>Semihosting . . . . .</b>	<b>.1049</b>
51.9.1	Guide Semihosting for IAR . . . . .	.1049
51.9.2	Guide Semihosting for Keil $\mu$ Vision . . . . .	.1049
51.9.3	Guide Semihosting for MCUXpresso IDE . . . . .	.1050
51.9.4	Guide Semihosting for ARMGCC . . . . .	.1051
<b>51.10</b>	<b>SWO . . . . .</b>	<b>.1053</b>
51.10.1	Guide SWO for SDK . . . . .	.1053
51.10.2	Guide SWO for Keil $\mu$ Vision . . . . .	.1054
51.10.3	Guide SWO for MCUXpresso IDE . . . . .	.1054
51.10.4	Guide SWO for ARMGCC . . . . .	.1054
<b>Chapter</b>	<b>Notification Framework</b>	
<b>52.1</b>	<b>Overview . . . . .</b>	<b>.1055</b>
<b>52.2</b>	<b>Notifier Overview . . . . .</b>	<b>.1055</b>
<b>52.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>.1057</b>
52.3.1	struct notifier_notification_block_t . . . . .	.1057
52.3.2	struct notifier_callback_config_t . . . . .	.1058
52.3.3	struct notifier_handle_t . . . . .	.1058
<b>52.4</b>	<b>Typedef Documentation . . . . .</b>	<b>.1059</b>
52.4.1	notifier_user_config_t . . . . .	.1059
52.4.2	notifier_user_function_t . . . . .	.1059
52.4.3	notifier_callback_t . . . . .	.1060



# Contents

Section Number	Title	Page Number
<b>52.5</b>	<b>Enumeration Type Documentation</b>	<b>1060</b>
52.5.1	_notifier_status	1060
52.5.2	notifier_policy_t	1061
52.5.3	notifier_notification_type_t	1061
52.5.4	notifier_callback_type_t	1061
<b>52.6</b>	<b>Function Documentation</b>	<b>1062</b>
52.6.1	NOTIFIER_CreateHandle	1062
52.6.2	NOTIFIER_SwitchConfig	1063
52.6.3	NOTIFIER_GetErrorCallbackIndex	1064
<b>Chapter Shell</b>		
<b>53.1</b>	<b>Overview</b>	<b>1065</b>
<b>53.2</b>	<b>Function groups</b>	<b>1065</b>
53.2.1	Initialization	1065
53.2.2	Advanced Feature	1065
53.2.3	Shell Operation	1066
<b>53.3</b>	<b>Data Structure Documentation</b>	<b>1067</b>
53.3.1	struct shell_context_struct	1067
53.3.2	struct shell_command_context_t	1068
53.3.3	struct shell_command_context_list_t	1068
<b>53.4</b>	<b>Macro Definition Documentation</b>	<b>1069</b>
53.4.1	SHELL_USE_HISTORY	1069
53.4.2	SHELL_SEARCH_IN_HIST	1069
53.4.3	SHELL_USE_FILE_STREAM	1069
53.4.4	SHELL_AUTO_COMPLETE	1069
53.4.5	SHELL_BUFFER_SIZE	1069
53.4.6	SHELL_MAX_ARGS	1069
53.4.7	SHELL_HIST_MAX	1069
53.4.8	SHELL_MAX_CMD	1069
<b>53.5</b>	<b>Typedef Documentation</b>	<b>1069</b>
53.5.1	send_data_cb_t	1069
53.5.2	recv_data_cb_t	1069
53.5.3	printf_data_t	1069
53.5.4	cmd_function_t	1069
<b>53.6</b>	<b>Enumeration Type Documentation</b>	<b>1069</b>
53.6.1	fun_key_status_t	1069
<b>53.7</b>	<b>Function Documentation</b>	<b>1070</b>
53.7.1	SHELL_Init	1070

# Contents

<b>Section Number</b>	<b>Title</b>	<b>Page Number</b>
53.7.2	SHELL_RegisterCommand . . . . .	.1070
53.7.3	SHELL_Main . . . . .	.1070
<b>Chapter Flexio_mculcd_edma</b>		
<b>54.1</b>	<b>Overview . . . . .</b>	<b>.1073</b>
<b>54.2</b>	<b>Data Structure Documentation . . . . .</b>	<b>.1073</b>
54.2.1	struct_flexio_mculcd_edma_handle . . . . .	.1073
<b>54.3</b>	<b>Macro Definition Documentation . . . . .</b>	<b>.1075</b>
54.3.1	FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION . . . . .	.1075
<b>54.4</b>	<b>Typedef Documentation . . . . .</b>	<b>.1075</b>
54.4.1	flexio_mculcd_edma_transfer_callback_t . . . . .	.1075
<b>54.5</b>	<b>Function Documentation . . . . .</b>	<b>.1075</b>
54.5.1	FLEXIO_MCULCD_TransferCreateHandleEDMA . . . . .	.1075
54.5.2	FLEXIO_MCULCD_TransferEDMA . . . . .	.1076
54.5.3	FLEXIO_MCULCD_TransferAbortEDMA . . . . .	.1076
54.5.4	FLEXIO_MCULCD_TransferGetCountEDMA . . . . .	.1077
<b>Chapter Data Structure Documentation</b>		
55.0.5	bee_region_config_t Struct Reference . . . . .	.1079

# Chapter 1

## Introduction

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

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

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

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

- IAR Embedded Workbench
- GNU Arm Embedded Toolchain

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

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

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

Table 1: MCUXpresso SDK Folder Structure

## Chapter 2

### Driver errors status

- `kStatus_CSI_NoEmptyBuffer` = 2900
- `kStatus_CSI_NoFullBuffer` = 2901
- `kStatus_CSI_QueueFull` = 2902
- `#kStatus_DCP_Again` = 6700
- `kStatus_EDMA_QueueFull` = 5100
- `kStatus_EDMA_Busy` = 5101
- `kStatus_ENET_RxFrameError` = 4000
- `kStatus_ENET_RxFrameFail` = 4001
- `kStatus_ENET_RxFrameEmpty` = 4002
- `kStatus_ENET_TxFrameOverLen` = 4003
- `kStatus_ENET_TxFrameBusy` = 4004
- `kStatus_ENET_TxFrameFail` = 4005
- `kStatus_ENET_PtpTsRingFull` = 4006
- `kStatus_ENET_PtpTsRingEmpty` = 4007
- `kStatus_FLEXCAN_TxBusy` = 5300
- `kStatus_FLEXCAN_TxIdle` = 5301
- `kStatus_FLEXCAN_TxSwitchToRx` = 5302
- `kStatus_FLEXCAN_RxBusy` = 5303
- `kStatus_FLEXCAN_RxIdle` = 5304
- `kStatus_FLEXCAN_RxOverflow` = 5305
- `kStatus_FLEXCAN_RxFifoBusy` = 5306
- `kStatus_FLEXCAN_RxFifoIdle` = 5307
- `kStatus_FLEXCAN_RxFifoOverflow` = 5308
- `kStatus_FLEXCAN_RxFifoWarning` = 5309
- `kStatus_FLEXCAN_ErrorStatus` = 5310
- `kStatus_FLEXCAN_UnHandled` = 5311
- `kStatus_FLEXIO_CAMERA_RxBusy` = 5500
- `kStatus_FLEXIO_CAMERA_RxIdle` = 5501
- `kStatus_FLEXIO_I2C_Busy` = 800
- `kStatus_FLEXIO_I2C_Idle` = 801
- `kStatus_FLEXIO_I2C_Nak` = 802
- `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_FLEXIO_MCULCD_Idle` = 2400
- `kStatus_FLEXIO_MCULCD_Busy` = 2401

- kStatus\_FLEXIO\_MCULCD\_Error = 2302
- kStatus\_FLEXIO\_SPI\_Busy = 501
- kStatus\_FLEXIO\_SPI\_Idle = 502
- kStatus\_FLEXIO\_SPI\_Error = 503
- kStatus\_FLEXIO\_UART\_TxBusy = 700
- kStatus\_FLEXIO\_UART\_RxBusy = 701
- kStatus\_FLEXIO\_UART\_TxIdle = 702
- kStatus\_FLEXIO\_UART\_RxIdle = 703
- kStatus\_FLEXIO\_UART\_ERROR = 704
- kStatus\_FLEXIO\_UART\_RxRingBufferOverrun = 705
- kStatus\_FLEXIO\_UART\_RxHardwareOverrun = 706
- #kStatus\_FLEXSPI\_Idle = 7000
- kStatus\_FLEXSPI\_Busy = 7001
- kStatus\_FLEXSPI\_SequenceExecutionTimeout = 7002
- kStatus\_FLEXSPI\_IpCommandSequenceError = 7003
- kStatus\_FLEXSPI\_IpCommandGrantTimeout = 7004
- kStatus\_LPI2C\_Busy = 900
- kStatus\_LPI2C\_Idle = 901
- kStatus\_LPI2C\_Nak = 902
- kStatus\_LPI2C\_FifoError = 903
- kStatus\_LPI2C\_BitError = 904
- kStatus\_LPI2C\_ArbitrationLost = 905
- kStatus\_LPI2C\_PinLowTimeout = 906
- kStatus\_LPI2C\_NoTransferInProgress = 907
- kStatus\_LPI2C\_DmaRequestFail = 908
- kStatus\_LPI2C\_Timeout = 909
- kStatus\_LPSPI\_Busy = 400
- kStatus\_LPSPI\_Error = 401
- kStatus\_LPSPI\_Idle = 402
- kStatus\_LPSPI\_OutOfRange = 403
- kStatus\_LPUART\_TxBusy = 1300
- kStatus\_LPUART\_RxBusy = 1301
- kStatus\_LPUART\_TxIdle = 1302
- kStatus\_LPUART\_RxIdle = 1303
- kStatus\_LPUART\_TxWatermarkTooLarge = 1304
- kStatus\_LPUART\_RxWatermarkTooLarge = 1305
- kStatus\_LPUART\_FlagCannotClearManually = 1306
- kStatus\_LPUART\_Error = 1307
- kStatus\_LPUART\_RxRingBufferOverrun = 1308
- kStatus\_LPUART\_RxHardwareOverrun = 1309
- kStatus\_LPUART\_NoiseError = 1310
- kStatus\_LPUART\_FramingError = 1311
- kStatus\_LPUART\_ParityError = 1312
- kStatus\_LPUART\_BaudrateNotSupport = 1313
- kStatus\_LPUART\_IdleLineDetected = 1314

- `kStatus_SAI_TxBusy` = 1900
- `kStatus_SAI_RxBusy` = 1901
- `kStatus_SAI_TxError` = 1902
- `kStatus_SAI_RxError` = 1903
- `kStatus_SAI_QueueFull` = 1904
- `kStatus_SAI_TxIdle` = 1905
- `kStatus_SAI_RxIdle` = 1906
- `#kStatus_SEMC_InvalidDeviceType` = 10000
- `#kStatus_SEMC_IpCommandExecutionError` = 10001
- `#kStatus_SEMC_AxiCommandExecutionError` = 10002
- `#kStatus_SEMC_InvalidMemorySize` = 10003
- `#kStatus_SEMC_InvalidIpcmdDataSize` = 10004
- `#kStatus_SEMC_InvalidAddressPortWidth` = 10005
- `#kStatus_SEMC_InvalidDataPortWidth` = 10006
- `#kStatus_SEMC_InvalidSwPinmuxSelection` = 10007
- `#kStatus_SEMC_InvalidBurstLength` = 10008
- `#kStatus_SEMC_InvalidColumnAddressBitWidth` = 10009
- `#kStatus_SEMC_InvalidBaseAddress` = 10010
- `#kStatus_SEMC_InvalidTimerSetting` = 10011
- `kStatus_SPDIF_RxDPLLLocked` = 7500
- `kStatus_SPDIF_TxFIFOError` = 7501
- `kStatus_SPDIF_TxFIFOResync` = 7502
- `kStatus_SPDIF_RxCnew` = 7503
- `kStatus_SPDIF_ValidatyNoGood` = 7504
- `kStatus_SPDIF_RxIllegalSymbol` = 7505
- `kStatus_SPDIF_RxParityBitError` = 7506
- `kStatus_SPDIF_UChannelOverrun` = 7507
- `kStatus_SPDIF_QChannelOverrun` = 7508
- `kStatus_SPDIF_UQChannelSync` = 7509
- `kStatus_SPDIF_UQChannelFrameError` = 7510
- `kStatus_SPDIF_RxFIFOError` = 7511
- `kStatus_SPDIF_RxFIFOResync` = 7512
- `kStatus_SPDIF_LockLoss` = 7513
- `kStatus_SPDIF_TxIdle` = 7514
- `kStatus_SPDIF_RxIdle` = 7515
- `kStatus_SPDIF_QueueFull` = 7516
- `kStatus_DMAMGR_ChannelOccupied` = 5200
- `kStatus_DMAMGR_ChannelNotUsed` = 5201
- `kStatus_DMAMGR_NoFreeChannel` = 5202
- `kStatus_NOTIFIER_ErrorNotificationBefore` = 9800
- `kStatus_NOTIFIER_ErrorNotificationAfter` = 9801





## Chapter 3 Architectural Overview

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

### Overview

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

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

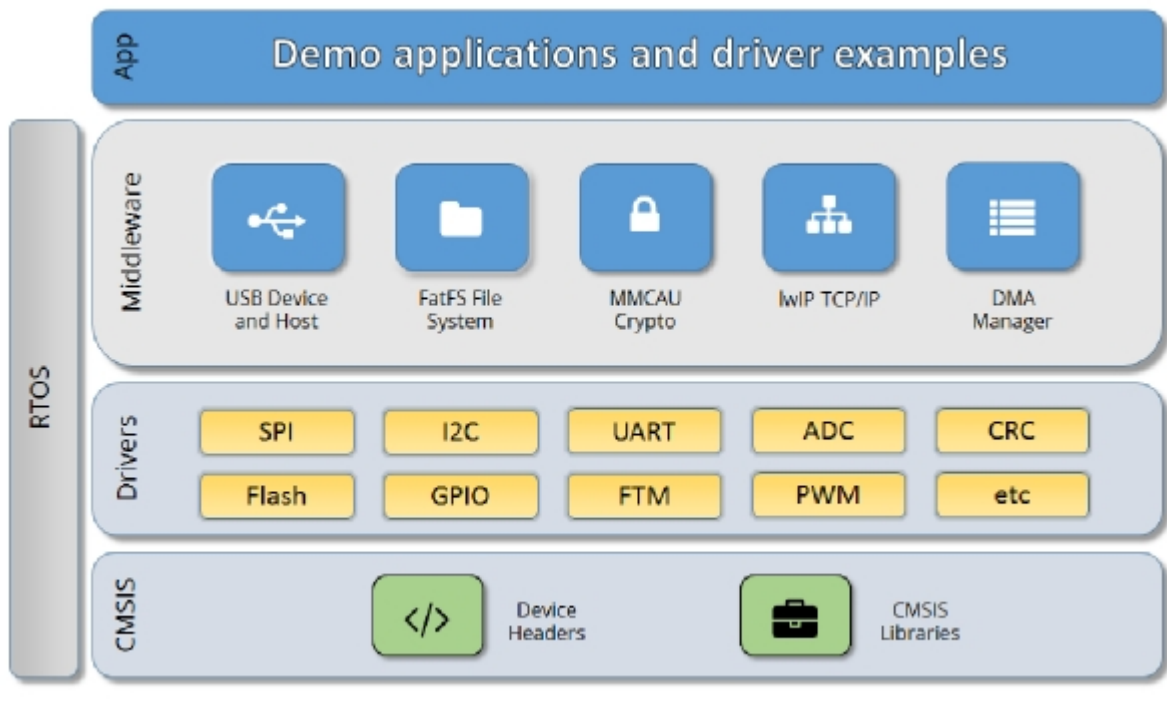


Figure 1: MCUXpresso SDK Block Diagram

### MCU header files

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

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

## CMSIS Support

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

## MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

## Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler
PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

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

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

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

### **Feature Header Files**

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

### **Application**

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



## Chapter 4 Trademarks

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

How to Reach Us:

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

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

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

NXP, the NXP logo, Freescale, the Freescale logo, Kinetis, and Processor Expert are trademarks of NXP B.V. Tower is a trademark of NXP B.V. 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.

© 2018 NXP B.V.



## Chapter 5

# ADC: 12-bit Analog to Digital Converter Driver

### 5.1 Overview

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

### 5.2 Typical use case

#### 5.2.1 Polling Configuration

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

#### 5.2.2 Polling Configuration

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

### Data Structures

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

### Macros

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

### Enumerations

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

## Typical use case

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



## Variables

- bool `adc_config_t::enableOverWrite`  
*Enable the overwriting.*
- bool `adc_config_t::enableContinuousConversion`  
*Enable the continuous conversion mode.*
- bool `adc_config_t::enableHighSpeed`  
*Enable the high-speed mode.*
- bool `adc_config_t::enableLowPower`  
*Enable the low power mode.*
- bool `adc_config_t::enableLongSample`  
*Enable the long sample mode.*
- bool `adc_config_t::enableAsynchronousClockOutput`  
*Enable the asynchronous clock output.*
- `adc_reference_voltage_source_t` `adc_config_t::referenceVoltageSource`  
*Select the reference voltage source.*
- `adc_sample_period_mode_t` `adc_config_t::samplePeriodMode`  
*Select the sample period in long sample mode or short mode.*
- `adc_clock_source_t` `adc_config_t::clockSource`  
*Select the input clock source to generate the internal clock ADCK.*
- `adc_clock_driver_t` `adc_config_t::clockDriver`  
*Select the divide ratio used by the ADC to generate the internal clock ADCK.*
- `adc_resolution_t` `adc_config_t::resolution`  
*Select the ADC resolution mode.*
- bool `adc_offset_config_t::enableSigned`  
*if false, The offset value is added with the raw result.*
- `uint32_t` `adc_offset_config_t::offsetValue`  
*User configurable offset value(0-4095).*
- `adc_hardware_compare_mode_t` `adc_hardware_compare_config_t::hardwareCompareMode`  
*Select the hardware compare mode.*
- `uint16_t` `adc_hardware_compare_config_t::value1`  
*Setting value1(0-4095) for hardware compare mode.*
- `uint16_t` `adc_hardware_compare_config_t::value2`  
*Setting value2(0-4095) for hardware compare mode.*
- `uint32_t` `adc_channel_config_t::channelNumber`  
*Setting the conversion channel number.*
- bool `adc_channel_config_t::enableInterruptOnConversionCompleted`  
*Generate an interrupt request once the conversion is completed.*

## Initialization

- void `ADC_Init` (`ADC_Type *base`, const `adc_config_t *config`)  
*Initialize the ADC module.*
- void `ADC_Deinit` (`ADC_Type *base`)  
*De-initializes the ADC module.*
- void `ADC_GetDefaultConfig` (`adc_config_t *config`)  
*Gets an available pre-defined settings for the converter's configuration.*
- void `ADC_SetChannelConfig` (`ADC_Type *base`, `uint32_t channelGroup`, const `adc_channel_config_t *config`)  
*Configures the conversion channel.*
- static `uint32_t` `ADC_GetChannelConversionValue` (`ADC_Type *base`, `uint32_t channelGroup`)  
*Gets the conversion value.*

## Data Structure Documentation

- static uint32\_t [ADC\\_GetChannelStatusFlags](#) (ADC\_Type \*base, uint32\_t channelGroup)  
*Gets the status flags of channel.*
- status\_t [ADC\\_DoAutoCalibration](#) (ADC\_Type \*base)  
*Automates the hardware calibration.*
- void [ADC\\_SetOffsetConfig](#) (ADC\_Type \*base, const [adc\\_offset\\_config\\_t](#) \*config)  
*Set user defined offset.*
- static void [ADC\\_EnableDMA](#) (ADC\_Type \*base, bool enable)  
*Enables generating the DMA trigger when the conversion is complete.*
- static void [ADC\\_EnableHardwareTrigger](#) (ADC\_Type \*base, bool enable)  
*Enables the hardware trigger mode.*
- void [ADC\\_SetHardwareCompareConfig](#) (ADC\_Type \*base, const [adc\\_hardware\\_compare\\_config\\_t](#) \*config)  
*Configures the hardware compare mode.*
- void [ADC\\_SetHardwareAverageConfig](#) (ADC\_Type \*base, [adc\\_hardware\\_average\\_mode\\_t](#) mode)  
*Configures the hardware average mode.*
- static uint32\_t [ADC\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Gets the converter's status flags.*
- void [ADC\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clears the converter's status flags.*

## 5.3 Data Structure Documentation

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

#### Data Fields

- bool [enableOverWrite](#)  
*Enable the overwriting.*
- bool [enableContinuousConversion](#)  
*Enable the continuous conversion mode.*
- bool [enableHighSpeed](#)  
*Enable the high-speed mode.*
- bool [enableLowPower](#)  
*Enable the low power mode.*
- bool [enableLongSample](#)  
*Enable the long sample mode.*
- bool [enableAsynchronousClockOutput](#)  
*Enable the asynchronous clock output.*
- [adc\\_reference\\_voltage\\_source\\_t](#) [referenceVoltageSource](#)  
*Select the reference voltage source.*
- [adc\\_sample\\_period\\_mode\\_t](#) [samplePeriodMode](#)  
*Select the sample period in long sample mode or short mode.*
- [adc\\_clock\\_source\\_t](#) [clockSource](#)  
*Select the input clock source to generate the internal clock ADCK.*
- [adc\\_clock\\_driver\\_t](#) [clockDriver](#)  
*Select the divide ratio used by the ADC to generate the internal clock ADCK.*
- [adc\\_resolution\\_t](#) [resolution](#)  
*Select the ADC resolution mode.*

### 5.3.2 struct adc\_offest\_config\_t

#### Data Fields

- bool [enableSigned](#)  
*if false, The offset value is added with the raw result.*
- uint32\_t [offsetValue](#)  
*User configurable offset value(0-4095).*

### 5.3.3 struct adc\_hardware\_compare\_config\_t

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

#### Data Fields

- [adc\\_hardware\\_compare\\_mode\\_t hardwareCompareMode](#)  
*Select the hardware compare mode.*
- uint16\_t [value1](#)  
*Setting value1(0-4095) for hardware compare mode.*
- uint16\_t [value2](#)  
*Setting value2(0-4095) for hardware compare mode.*

### 5.3.4 struct adc\_channel\_config\_t

#### Data Fields

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

## 5.4 Macro Definition Documentation

### 5.4.1 #define FSL\_ADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

## Enumeration Type Documentation

### 5.5 Enumeration Type Documentation

#### 5.5.1 enum adc\_status\_flags\_t

Enumerator

- kADC\_ConversionActiveFlag* Conversion is active,not support w1c.
- kADC\_CalibrationFailedFlag* Calibration is failed,support w1c.
- kADC\_Asynchronous WakeupInterruptFlag* Asynchronous wakeup interrupt occurred, support w1c.

#### 5.5.2 enum adc\_reference\_voltage\_source\_t

Enumerator

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

#### 5.5.3 enum adc\_sample\_period\_mode\_t

Enumerator

- kADC\_SamplePeriod2or12Clocks* Long sample 12 clocks or short sample 2 clocks.
- kADC\_SamplePeriod4or16Clocks* Long sample 16 clocks or short sample 4 clocks.
- kADC\_SamplePeriod6or20Clocks* Long sample 20 clocks or short sample 6 clocks.
- kADC\_SamplePeriod8or24Clocks* Long sample 24 clocks or short sample 8 clocks.
- kADC\_SamplePeriodLong12Clcoks* Long sample 12 clocks.
- kADC\_SamplePeriodLong16Clcoks* Long sample 16 clocks.
- kADC\_SamplePeriodLong20Clcoks* Long sample 20 clocks.
- kADC\_SamplePeriodLong24Clcoks* Long sample 24 clocks.
- kADC\_SamplePeriodShort2Clocks* Short sample 2 clocks.
- kADC\_SamplePeriodShort4Clocks* Short sample 4 clocks.
- kADC\_SamplePeriodShort6Clocks* Short sample 6 clocks.
- kADC\_SamplePeriodShort8Clocks* Short sample 8 clocks.

#### 5.5.4 enum adc\_clock\_source\_t

Enumerator

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

### 5.5.5 enum adc\_clock\_driver\_t

Enumerator

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

### 5.5.6 enum adc\_resolution\_t

Enumerator

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

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

Enumerator

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

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

Enumerator

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

## 5.6 Function Documentation

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

## Function Documentation

### Parameters

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

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

#### Parameters

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

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

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

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

#### Parameters

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

### 5.6.4 void ADC\_SetChannelConfig ( ADC\_Type \* *base*, uint32\_t *channelGroup*, const adc\_channel\_config\_t \* *config* )

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

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC has more than one group of status and control registers, one for each conversion. The channel group parameter indicates which group of registers are used, for

example channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any point, only one of the channel groups is actively controlling ADC conversions. The channel group 0 is used for both software and hardware trigger modes. Channel groups 1 and greater indicate potentially multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the appropriate MCU reference manual about the number of SC1n registers (channel groups) specific to this device. None of the channel groups 1 or greater are used for software trigger operation. Therefore, writing to these channel groups does not initiate a new conversion. Updating the channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

Parameters

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

### 5.6.5 `static uint32_t ADC_GetChannelConversionValue ( ADC_Type * base, uint32_t channelGroup ) [inline], [static]`

Parameters

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

Returns

Conversion value.

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

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

## Function Documentation

### Parameters

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

### Returns

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

### 5.6.7 **status\_t** ADC\_DoAutoCalibration ( ADC\_Type \* *base* )

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

### Parameters

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

### Returns

Execution status.

### Return values

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

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

### Parameters

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

### 5.6.9 **static void** ADC\_EnabledDMA ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]



Parameters

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

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

Parameters

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

#### 5.6.11 void ADC\_SetHardwareCompareConfig ( ADC\_Type \* *base*, const adc\_hardware\_compare\_config\_t \* *config* )

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

Parameters

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

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

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

Parameters

## Function Documentation

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

### 5.6.13 `static uint32_t ADC_GetStatusFlags ( ADC_Type * base ) [inline], [static]`

#### Parameters

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

#### Returns

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

### 5.6.14 `void ADC_ClearStatusFlags ( ADC_Type * base, uint32_t mask )`

#### Parameters

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

## 5.7 Variable Documentation

5.7.1 **bool** `adc_config_t::enableOverWrite`

5.7.2 **bool** `adc_config_t::enableContinuousConversion`

5.7.3 **bool** `adc_config_t::enableHighSpeed`

5.7.4 **bool** `adc_config_t::enableLowPower`

5.7.5 **bool** `adc_config_t::enableLongSample`

5.7.6 **bool** `adc_config_t::enableAsynchronousClockOutput`

5.7.7 **adc\_reference\_voltage\_source\_t** `adc_config_t::referenceVoltageSource`

5.7.8 **adc\_sample\_period\_mode\_t** `adc_config_t::samplePeriodMode`

5.7.9 **adc\_clock\_source\_t** `adc_config_t::clockSource`

5.7.10 **adc\_clock\_driver\_t** `adc_config_t::clockDriver`

5.7.11 **adc\_resolution\_t** `adc_config_t::resolution`

5.7.12 **bool** `adc_offset_config_t::enableSigned`

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

5.7.13 **uint32\_t** `adc_offset_config_t::offsetValue`

5.7.14 **adc\_hardware\_compare\_mode\_t** `adc_hardware_compare_config_t::hardwareCompareMode`

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

## Variable Documentation

**5.7.15** `uint16_t adc_hardware_compare_config_t::value1`

**5.7.16** `uint16_t adc_hardware_compare_config_t::value2`

**5.7.17** `uint32_t adc_channel_config_t::channelNumber`

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

**5.7.18** `bool adc_channel_config_t::enableInterruptOnConversionCompleted`

## Chapter 6

# ADC\_ETC: ADC External Trigger Control

### 6.1 Overview

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

### 6.2 Typical use case

#### 6.2.1 Software trigger Configuration

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

#### 6.2.2 Hardware trigger Configuration

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

### Data Structures

- struct [adc\\_etc\\_config\\_t](#)  
*ADC\_ETC configuration. [More...](#)*
- struct [adc\\_etc\\_trigger\\_chain\\_config\\_t](#)  
*ADC\_ETC trigger chain configuration. [More...](#)*
- struct [adc\\_etc\\_trigger\\_config\\_t](#)  
*ADC\_ETC trigger configuration. [More...](#)*

### Macros

- #define [FSL\\_ADC\\_ETC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*ADC\_ETC driver version.*
- #define [ADC\\_ETC\\_DMA\\_CTRL\\_TRGn\\_REQ\\_MASK](#) 0xFF0000U  
*The mask of status flags cleared by writing 1.*

### Enumerations

- enum [\\_adc\\_etc\\_status\\_flag\\_mask](#)  
*ADC\_ETC customized status flags mask.*
- enum [adc\\_etc\\_external\\_trigger\\_source\\_t](#)  
*External triggers sources.*
- enum [adc\\_etc\\_interrupt\\_enable\\_t](#)  
*Interrupt enable/disable mask.*

## Function Documentation

### Initialization

- void `ADC_ETC_Init` (`ADC_ETC_Type *base`, const `adc_etc_config_t *config`)  
*Initialize the ADC\_ETC module.*
- void `ADC_ETC_Deinit` (`ADC_ETC_Type *base`)  
*De-Initialize the ADC\_ETC module.*

### 6.3 Data Structure Documentation

#### 6.3.1 struct `adc_etc_config_t`

#### 6.3.2 struct `adc_etc_trigger_chain_config_t`

#### 6.3.3 struct `adc_etc_trigger_config_t`

### 6.4 Macro Definition Documentation

#### 6.4.1 `#define FSL_ADC_ETC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

Version 2.0.0.

#### 6.4.2 `#define ADC_ETC_DMA_CTRL_TRGn_REQ_MASK 0xFF0000U`

### 6.5 Function Documentation

#### 6.5.1 void `ADC_ETC_Init` ( `ADC_ETC_Type * base`, const `adc_etc_config_t * config` )

Parameters

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

#### 6.5.2 void `ADC_ETC_Deinit` ( `ADC_ETC_Type * base` )

Parameters

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





# Chapter 7

## AIPSTZ: AHB to IP Bridge

### 7.1 Overview

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

### Enumerations

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

### Driver version

- #define `FSL_AIPSTZ_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
    *Version 2.0.0.*

### Initialization and deinitialization

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

## 7.2 Enumeration Type Documentation

### 7.2.1 enum `aipstz_master_privilege_level_t`

Enumerator

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

## Function Documentation

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

### 7.2.2 enum aipstz\_master\_t

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

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

### 7.2.4 enum aipstz\_peripheral\_t

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

## 7.3 Function Documentation

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

Parameters

<i>base</i>	AIPSTZ peripheral base pointer
<i>master</i>	Masters for AIPSTZ.
<i>privilegeConfig</i>	Configuration is ORed from .

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

Parameters

<i>base</i>	AIPSTZ peripheral base pointer
<i>master</i>	Peripheral for AIPSTZ.

<i>accessControl</i>	Configuration is ORed from .
----------------------	------------------------------



## Chapter 8

# AOI: Crossbar AND/OR/INVERT Driver

### 8.1 Overview

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

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

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

### 8.2 Function groups

#### 8.2.1 AOI Initialization

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

See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/aoi## A-OI Get Set Operation {#AOIOperation}` The AOI module provides a universal boolean function generator using a four-term sum of products expression with each product term containing true or complement values of the four selected event inputs (A, B, C, D). The AOI is a highly programmable module for creating combinational boolean outputs for use as hardware triggers. Each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. To configure the selected AOI module event, call the API of the [AOI\\_SetEventLogicConfig\(\)](#) function. To get the current event state configure, call the API of [AOI\\_GetEventLogicConfig\(\)](#) function. The AOI module does not support any special modes of operation. See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/aoi# Typical use case {#AOIUsercase}` The A-OI module is designed to be integrated in conjunction with one or more inter-peripheral crossbar switch (XBAR) modules. A crossbar switch is typically used to select the 4\*n AOI inputs from among available peripheral outputs and GPIO signals. The n EVENTn outputs from the AOI module are typically used as additional inputs to a second crossbar switch, adding to it the ability to connect to its outputs an arbitrary 4-input boolean function of its other inputs.

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

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

## Function groups

## Data Structures

- struct [aoi\\_event\\_config\\_t](#)  
*AOI event configuration structure. [More...](#)*

## Macros

- #define [AOI](#) AOI0  
*AOI peripheral address.*

## Enumerations

- enum [aoi\\_input\\_config\\_t](#) {  
    [kAOI\\_LogicZero](#) = 0x0U,  
    [kAOI\\_InputSignal](#) = 0x1U,  
    [kAOI\\_InvInputSignal](#) = 0x2U,  
    [kAOI\\_LogicOne](#) = 0x3U }  
*AOI input configurations.*
- enum [aoi\\_event\\_t](#) {  
    [kAOI\\_Event0](#) = 0x0U,  
    [kAOI\\_Event1](#) = 0x1U,  
    [kAOI\\_Event2](#) = 0x2U,  
    [kAOI\\_Event3](#) = 0x3U }  
*AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).*

## Driver version

- #define [FSL\\_AOI\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## AOI Initialization

- void [AOI\\_Init](#) (AOI\_Type \*base)  
*Initializes an AOI instance for operation.*
- void [AOI\\_Deinit](#) (AOI\_Type \*base)  
*Deinitializes an AOI instance for operation.*

## AOI Get Set Operation

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

## 8.3 Data Structure Documentation

### 8.3.1 struct aoi\_event\_config\_t

Defines structure `_aoi_event_config` and use the `AOI_SetEventLogicConfig()` function to make whole event configuration.

#### Data Fields

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

## 8.4 Macro Definition Documentation

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

## Function Documentation

### 8.5 Enumeration Type Documentation

#### 8.5.1 enum aoi\_input\_config\_t

The selection item represents the Boolean evaluations.

Enumerator

*kAOI\_LogicZero* Forces the input to logical zero.

*kAOI\_InputSignal* Passes the input signal.

*kAOI\_InvInputSignal* Inverts the input signal.

*kAOI\_LogicOne* Forces the input to logical one.

#### 8.5.2 enum aoi\_event\_t

Enumerator

*kAOI\_Event0* Event 0 index.

*kAOI\_Event1* Event 1 index.

*kAOI\_Event2* Event 2 index.

*kAOI\_Event3* Event 3 index.

### 8.6 Function Documentation

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

This function un-gates the AOI clock.

Parameters

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

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

This function shutdowns AOI module.

Parameters

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

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

This function returns the Boolean evaluation associated.



Example:

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

Parameters

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

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

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

Example:

```
aoi_event_config_t demoEventLogicStruct;

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

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

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

demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
demoEventLogicStruct.PT3DC = kAOI_LogicOne;

AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

Parameters

---

## Function Documentation

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

## Chapter 9

# BEE: Bus Encryption Engine

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

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

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

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

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

### 9.1 BEE Driver Initialization and Configuration

The function `BEE_Init()` initializes the BEE to default values. The function `BEE_GetDefaultConfig()` loads default values to the BEE region configuration structure. The default values are described below.

See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/bee`. The function `BEE_Deinit()` performs a hardware reset of BEE module and disables clocks. Configuration and keys from software for both regions are cleared.

### 9.2 Enable & Disable BEE

The function `BEE_Enable()` enables decryption using BEE. The function `BEE_Disable()` disables decryption using BEE.

### 9.3 Set BEE region config and key

The function `BEE_SetRegionConfig()` sets BEE region settings according to given configuration structure. The structure is described below.

See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/bee`. The function `BEE_SetRegionKey()` loads given AES key and nonce (only AES CTR mode) to BEE register for the given region. Note that eFuse `BEE_KEYX_SEL` must be set accordingly to be able to load and use the key loaded in BEE registers. Otherwise, the key cannot be loaded and BEE uses the key from `OTPMK` or `SW_GP2`.

### 9.4 Status

Provides functions to get and clear the BEE status.

## Get offset

The function `BEE_GetStatusFlags()` returns status of BEE peripheral. The function `BEE_ClearStatusFlags()` clears the BEE status flags.

## 9.5 Get offset

The function `BEE_GetOffset()` calculates offset that must be set for BEE region to access physical memory location.

### 9.5.1 BEE example

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

# Chapter 10

## CACHE: CACHE Memory Controller

### 10.1 Overview

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

The CACHE driver is created to help the user more easily operate the cache memory. The APIs for basic operations are including the following three levels: 1L. The L1 cache driver API. This level provides the level 1 caches controller drivers. The L1 caches are mainly integrated in the Core memory system, Cortex-M7 L1 caches, etc. For our Cortex-M4 series platforms, the L1 cache is the local memory controller (LMEM) which is not integrated in the Cortex-M4 processor memory system.

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

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

### 10.2 Function groups

#### L1 CACHE Operation {#L1CACHE MaintainOperation}

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

#### L2 CACHE Operation {#L2CACHE MaintainOperation}

The L2 CACHE does not divide the cache to data and code. Instead, this function group provides one group cache maintenance operations as Enable/Disable/Invalidate/Clean/CleanInvalidate by all and by address range. Except the maintenance operation APIs, the L2 CACHE has its initialization/configure API. The user can use the default configure parameter by calling L2CACHE\_GetDefaultConfig() or changing the parameters as they wish. Then, call L2CACHE\_Init to do the L2 CACHE initialization. After initialization, the L2 cache can then be enabled.

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

## Macro Definition Documentation

### Driver version

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

### Control for cortex-m7 L1 cache

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

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

Mainly used for many drivers for easy cache operation.

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

## 10.3 Macro Definition Documentation

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

## 10.4 Function Documentation

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

## Function Documentation

### Parameters

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

### Note

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

### 10.4.2 `static void L1CACHE_InvalidateDCacheByRange ( uint32_t address, uint32_t size_byte ) [inline], [static]`

### Parameters

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

### Note

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

### 10.4.3 `static void L1CACHE_CleanDCacheByRange ( uint32_t address, uint32_t size_byte ) [inline], [static]`

### Parameters

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

### Note

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



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

## Function Documentation

### Parameters

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

### Note

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

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

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

### Parameters

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

### Note

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

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

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

### Parameters

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

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

Note

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

**10.4.7 void DCACHE\_CleanByRange ( uint32\_t address, uint32\_t size\_byte )**

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

Parameters

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

Note

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

**10.4.8 void DCACHE\_CleanInvalidateByRange ( uint32\_t address, uint32\_t size\_byte )**

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

Parameters

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

Note

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



# Chapter 11

## CMP: Analog Comparator Driver

### 11.1 Overview

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

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

### 11.2 Typical use case

#### 11.2.1 Polling Configuration

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

#### 11.2.2 Interrupt Configuration

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

### Data Structures

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

### Enumerations

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

## Typical use case

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

## Driver version

- #define `FSL_CMP_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 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_EnableDMA` (CMP\_Type \*base, bool enable)  
    *Enables/disables the DMA request for rising/falling events.*
- static void `CMP_EnableWindowMode` (CMP\_Type \*base, bool enable)  
    *Enables/disables the window mode.*
- void `CMP_SetFilterConfig` (CMP\_Type \*base, const `cmp_filter_config_t` \*config)  
    *Configures the filter.*
- void `CMP_SetDACConfig` (CMP\_Type \*base, const `cmp_dac_config_t` \*config)  
    *Configures the internal DAC.*
- void `CMP_EnableInterrupts` (CMP\_Type \*base, uint32\_t mask)  
    *Enables the interrupts.*
- void `CMP_DisableInterrupts` (CMP\_Type \*base, uint32\_t mask)  
    *Disables the interrupts.*

## Results

- uint32\_t `CMP_GetStatusFlags` (CMP\_Type \*base)  
    *Gets the status flags.*
- void `CMP_ClearStatusFlags` (CMP\_Type \*base, uint32\_t mask)  
    *Clears the status flags.*

## 11.3 Data Structure Documentation

### 11.3.1 struct cmp\_config\_t

#### Data Fields

- bool [enableCmp](#)  
*Enable the CMP module.*
- [cmp\\_hysteresis\\_mode\\_t](#) [hysteresisMode](#)  
*CMP Hysteresis mode.*
- bool [enableHighSpeed](#)  
*Enable High-speed (HS) comparison mode.*
- bool [enableInvertOutput](#)  
*Enable the inverted comparator output.*
- bool [useUnfilteredOutput](#)  
*Set the compare output(COUT) to equal COUTA(true) or COUT(false).*
- bool [enablePinOut](#)  
*The comparator output is available on the associated pin.*

#### 11.3.1.0.0.1 Field Documentation

11.3.1.0.0.1.1 **bool** `cmp_config_t::enableCmp`

11.3.1.0.0.1.2 **cmp\_hysteresis\_mode\_t** `cmp_config_t::hysteresisMode`

11.3.1.0.0.1.3 **bool** `cmp_config_t::enableHighSpeed`

11.3.1.0.0.1.4 **bool** `cmp_config_t::enableInvertOutput`

11.3.1.0.0.1.5 **bool** `cmp_config_t::useUnfilteredOutput`

11.3.1.0.0.1.6 **bool** `cmp_config_t::enablePinOut`

### 11.3.2 struct cmp\_filter\_config\_t

#### Data Fields

- bool [enableSample](#)  
*Using the external SAMPLE as a sampling clock input or using a divided bus clock.*
- [uint8\\_t](#) [filterCount](#)  
*Filter Sample Count.*
- [uint8\\_t](#) [filterPeriod](#)  
*Filter Sample Period.*

## Enumeration Type Documentation

### 11.3.2.0.0.2 Field Documentation

#### 11.3.2.0.0.2.1 `bool cmp_filter_config_t::enableSample`

#### 11.3.2.0.0.2.2 `uint8_t cmp_filter_config_t::filterCount`

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

#### 11.3.2.0.0.2.3 `uint8_t cmp_filter_config_t::filterPeriod`

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

### 11.3.3 `struct cmp_dac_config_t`

#### Data Fields

- [`cmp\_reference\_voltage\_source\_t referenceVoltageSource`](#)  
*Supply voltage reference source.*
- `uint8_t DACValue`  
*Value for the DAC Output Voltage.*

#### 11.3.3.0.0.3 Field Documentation

##### 11.3.3.0.0.3.1 `cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource`

##### 11.3.3.0.0.3.2 `uint8_t cmp_dac_config_t::DACValue`

Available range is 0-63.

## 11.4 Macro Definition Documentation

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

## 11.5 Enumeration Type Documentation

### 11.5.1 `enum _cmp_interrupt_enable`

Enumerator

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

### 11.5.2 `enum _cmp_status_flags`

Enumerator

- `kCMP_OutputRisingEventFlag`* Rising-edge on the comparison output has occurred.



***kCMP\_OutputFallingEventFlag*** Falling-edge on the comparison output has occurred.  
***kCMP\_OutputAssertEventFlag*** Return the current value of the analog comparator output.

### 11.5.3 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.

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

Enumerator

***kCMP\_VrefSourceVin1*** Vin1 is selected as a resistor ladder network supply reference Vin.  
***kCMP\_VrefSourceVin2*** Vin2 is selected as a resistor ladder network supply reference Vin.

## 11.6 Function Documentation

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

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

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

Parameters

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

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

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

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

## Function Documentation

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

Parameters

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

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

Parameters

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

### 11.6.4 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.
---------------	-----------------------------------------

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

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

Parameters

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

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

## Function Documentation

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

Returns

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

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

## Function Documentation

### Parameters

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

## Chapter 12

# CSI: CMOS Sensor Interface

### 12.1 Overview

The MCUXpresso SDK provides a driver for the CMOS Sensor Interface (CSI)

The CSI enables the chip to connect directly to external CMOS image sensors. The CSI driver provides functional APIs and transactional APIs for the CSI module. The functional APIs implement the basic functions, so the user can construct them for a special use case. The transactional APIs provide a queue mechanism in order for the user to submit an empty frame buffer and get a fully-filled frame buffer easily.

### 12.2 Frame Buffer Queue

The CSI transactional functions maintain a frame buffer queue. The queue size is defined by the macro `CSI_DRIVER_QUEUE_SIZE`. The queue size is 4 by default, but the user can override it by redefining the macro value in the project setting.

To use transactional APIs, first call [CSI\\_TransferCreateHandle](#) to create a handle to save the CSI driver state. This function initializes the frame buffer queue to empty status.

After the handle is created, the function [CSI\\_TransferSubmitEmptyBuffer](#) can be used to submit the empty frame buffer to the queue. If the queue does not have room to save the new empty frame buffers, this function returns with an error. It is not necessary to check the queue rooms before submitting an empty frame buffer. After this step, the application can call [CSI\\_TransferStart](#) to start the transfer. There must be at least two empty buffers in the queue, otherwise this function returns an error. The incoming frames are saved to the empty buffers one by one, and a callback is provided when every frame completed. To get the fully-filled frame buffer, call the function [CSI\\_TransferGetFullBuffer](#). This function returns an error if the frame buffer queue does not have full buffers. Therefore, it is not necessary to check the full buffer number in the queue before this function.

To stop the transfer, call the function [CSI\\_TransferStop](#) at anytime. If the queue has some full frame buffers, the application can still read them out after this stop function.

During the transfer, if all empty buffers are fully-filled, the CSI module will be stopped silently. Be aware that the stop here is different with the stop by [CSI\\_TransferStop](#). When the application submits new buffers to the queue and the queue has more than two empty buffers, the CSI module starts automatically to same frame to the empty buffer.

### 12.3 Typical use case

#### 12.3.1 Receive with functional APIs

In this example, the CSI is enabled to save 10 frames.

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

## Typical use case

### 12.3.2 Receive with transactional APIs

This example shows how to receive the frame using CSI driver transactional APIs.

See the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/csi. In this example, there are only 3 buffers. The CSI driver queue size is 4, so it is not necessary to check queue room before submit the empty buffer.

## Data Structures

- struct `csi_config_t`  
*Configuration to initialize the CSI module. [More...](#)*
- struct `csi_handle_t`  
*CSI handle structure. [More...](#)*

## Macros

- #define `CSI_DRIVER_QUEUE_SIZE` 4U  
*Size of the frame buffer queue used in CSI transactional function.*

## Typedefs

- typedef void(\* `csi_transfer_callback_t`)(CSI\_Type \*base, csi\_handle\_t \*handle, status\_t status, void \*userData)  
*CSI transfer callback function.*

## Enumerations

- enum `_csi_status` {  
    `kStatus_CSI_NoEmptyBuffer` = MAKE\_STATUS(kStatusGroup\_CSI, 0),  
    `kStatus_CSI_NoFullBuffer` = MAKE\_STATUS(kStatusGroup\_CSI, 1),  
    `kStatus_CSI_QueueFull` = MAKE\_STATUS(kStatusGroup\_CSI, 2),  
    `kStatus_CSI_FrameDone` = MAKE\_STATUS(kStatusGroup\_CSI, 3) }  
*Error codes for the CSI driver.*
- enum `csi_work_mode_t` {  
    `kCSI_GatedClockMode` = CSI\_CSICR1\_GCLK\_MODE(1U),  
    `kCSI_NonGatedClockMode` = 0U,  
    `kCSI_CCIR656ProgressiveMode` = CSI\_CSICR1\_CCIR\_EN(1U) }  
*CSI work mode.*
- enum `csi_data_bus_t` { `kCSI_DataBus8Bit` }
- enum `_csi_polarity_flags` {  
    `kCSI_HsyncActiveLow` = 0U,  
    `kCSI_HsyncActiveHigh` = CSI\_CSICR1\_HSYNC\_POL\_MASK,  
    `kCSI_DataLatchOnRisingEdge` = CSI\_CSICR1\_REEDGE\_MASK,  
    `kCSI_DataLatchOnFallingEdge` = 0U,  
    `kCSI_VsyncActiveHigh` = 0U,  
    `kCSI_VsyncActiveLow` = CSI\_CSICR1\_SOF\_POL\_MASK }



- CSI signal polarity.*

```
enum csi_fifo_t {
    kCSI_RxFifo = (1U << 0U),
    kCSI_StatFifo = (1U << 1U),
    kCSI_AllFifo = 0x01 | 0x02 }

```

*The CSI FIFO, used for FIFO operation.*
- ```
enum _csi_interrupt_enable {
    kCSI_EndOfFrameInterruptEnable = CSI_CSICR1_EOF_INT_EN_MASK,
    kCSI_ChangeOfFieldInterruptEnable = CSI_CSICR1_COF_INT_EN_MASK,
    kCSI_StatFifoOverrunInterruptEnable = CSI_CSICR1_SF_OR_INTEN_MASK,
    kCSI_RxFifoOverrunInterruptEnable = CSI_CSICR1_RF_OR_INTEN_MASK,
    kCSI_StatFifoDmaDoneInterruptEnable,
    kCSI_StatFifoFullInterruptEnable = CSI_CSICR1_STATFF_INTEN_MASK,
    kCSI_RxBuffer1DmaDoneInterruptEnable,
    kCSI_RxBuffer0DmaDoneInterruptEnable,
    kCSI_RxFifoFullInterruptEnable = CSI_CSICR1_RXFF_INTEN_MASK,
    kCSI_StartOfFrameInterruptEnable = CSI_CSICR1_SOF_INTEN_MASK,
    kCSI_EccErrorInterruptEnable = CSI_CSICR3_ECC_INT_EN_MASK,
    kCSI_AhbResErrorInterruptEnable = CSI_CSICR3_HRESP_ERR_EN_MASK,
    kCSI_BaseAddrChangeErrorInterruptEnable = CSI_CSICR18_BASEADDR_CHANGE_ERROR-
        _IE_MASK << 6U,
    kCSI_Field0DoneInterruptEnable = CSI_CSICR18_FIELD0_DONE_IE_MASK << 6U,
    kCSI_Field1DoneInterruptEnable = CSI_CSICR18_DMA_FIELD1_DONE_IE_MASK << 6U }

```

*CSI feature interrupt source.*
- ```
enum _csi_flags {
    kCSI_RxFifoDataReadyFlag = CSI_CSISR_DRDY_MASK,
    kCSI_EccErrorFlag = CSI_CSISR_ECC_INT_MASK,
    kCSI_AhbResErrorFlag = CSI_CSISR_HRESP_ERR_INT_MASK,
    kCSI_ChangeOfFieldFlag = CSI_CSISR_COF_INT_MASK,
    kCSI_Field0PresentFlag = CSI_CSISR_F1_INT_MASK,
    kCSI_Field1PresentFlag = CSI_CSISR_F2_INT_MASK,
    kCSI_StartOfFrameFlag = CSI_CSISR_SOF_INT_MASK,
    kCSI_EndOfFrameFlag = CSI_CSISR_EOF_INT_MASK,
    kCSI_RxFifoFullFlag = CSI_CSISR_RxFF_INT_MASK,
    kCSI_RxBuffer1DmaDoneFlag = CSI_CSISR_DMA_TSF_DONE_FB2_MASK,
    kCSI_RxBuffer0DmaDoneFlag = CSI_CSISR_DMA_TSF_DONE_FB1_MASK,
    kCSI_StatFifoFullFlag = CSI_CSISR_STATFF_INT_MASK,
    kCSI_StatFifoDmaDoneFlag = CSI_CSISR_DMA_TSF_DONE_SFF_MASK,
    kCSI_StatFifoOverrunFlag = CSI_CSISR_SF_OR_INT_MASK,
    kCSI_RxFifoOverrunFlag = CSI_CSISR_RF_OR_INT_MASK,
    kCSI_Field0DoneFlag = CSI_CSISR_DMA_FIELD0_DONE_MASK,
    kCSI_Field1DoneFlag = CSI_CSISR_DMA_FIELD1_DONE_MASK,
    kCSI_BaseAddrChangeErrorFlag = CSI_CSISR_BASEADDR_CHHANGE_ERROR_MASK }

```

*CSI status flags.*

## Typical use case

## Driver version

- #define **FSL\_CSI\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 0))

## Initialization and deinitialization

- status\_t **CSI\_Init** (CSI\_Type \*base, const **csi\_config\_t** \*config)  
*Initialize the CSI.*
- void **CSI\_Deinit** (CSI\_Type \*base)  
*De-initialize the CSI.*
- void **CSI\_Reset** (CSI\_Type \*base)  
*Reset the CSI.*
- void **CSI\_GetDefaultConfig** (**csi\_config\_t** \*config)  
*Get the default configuration for to initialize the CSI.*

## Module operation

- void **CSI\_ClearFifo** (CSI\_Type \*base, **csi\_fifo\_t** fifo)  
*Clear the CSI FIFO.*
- void **CSI\_ReflashFifoDma** (CSI\_Type \*base, **csi\_fifo\_t** fifo)  
*Reflash the CSI FIFO DMA.*
- void **CSI\_EnableFifoDmaRequest** (CSI\_Type \*base, **csi\_fifo\_t** fifo, bool enable)  
*Enable or disable the CSI FIFO DMA request.*
- static void **CSI\_Start** (CSI\_Type \*base)  
*Start to receive data.*
- static void **CSI\_Stop** (CSI\_Type \*base)  
*Stop to receiving data.*
- void **CSI\_SetRxBufferAddr** (CSI\_Type \*base, uint8\_t index, uint32\_t addr)  
*Set the RX frame buffer address.*

## Interrupts

- void **CSI\_EnableInterrupts** (CSI\_Type \*base, uint32\_t mask)  
*Enables CSI interrupt requests.*
- void **CSI\_DisableInterrupts** (CSI\_Type \*base, uint32\_t mask)  
*Disable CSI interrupt requests.*

## Status

- static uint32\_t **CSI\_GetStatusFlags** (CSI\_Type \*base)  
*Gets the CSI status flags.*
- static void **CSI\_ClearStatusFlags** (CSI\_Type \*base, uint32\_t statusMask)  
*Clears the CSI status flag.*

## Transactional

- status\_t **CSI\_TransferCreateHandle** (CSI\_Type \*base, **csi\_handle\_t** \*handle, **csi\_transfer\_callback\_t** callback, void \*userData)  
*Initializes the CSI handle.*
- status\_t **CSI\_TransferStart** (CSI\_Type \*base, **csi\_handle\_t** \*handle)  
*Start the transfer using transactional functions.*

- status\_t [CSI\\_TransferStop](#) (CSI\_Type \*base, csi\_handle\_t \*handle)  
*Stop the transfer using transactional functions.*
- status\_t [CSI\\_TransferSubmitEmptyBuffer](#) (CSI\_Type \*base, csi\_handle\_t \*handle, uint32\_t frame-Buffer)  
*Submit empty frame buffer to queue.*
- status\_t [CSI\\_TransferGetFullBuffer](#) (CSI\_Type \*base, csi\_handle\_t \*handle, uint32\_t \*frame-Buffer)  
*Get one full frame buffer from queue.*
- void [CSI\\_TransferHandleIRQ](#) (CSI\_Type \*base, csi\_handle\_t \*handle)  
*CSI IRQ handle function.*

## 12.4 Data Structure Documentation

### 12.4.1 struct csi\_config\_t

#### Data Fields

- uint16\_t [width](#)  
*Pixels of the input frame.*
- uint16\_t [height](#)  
*Lines of the input frame.*
- uint32\_t [polarityFlags](#)  
*Timing signal polarity flags, OR'ed value of [\\_csi\\_polarity\\_flags](#).*
- uint8\_t [bytesPerPixel](#)  
*Bytes per pixel, valid values are:*
- uint16\_t [linePitch\\_Bytes](#)  
*Frame buffer line pitch, must be 8-byte aligned.*
- [csi\\_work\\_mode\\_t](#) [workMode](#)  
*CSI work mode.*
- [csi\\_data\\_bus\\_t](#) [dataBus](#)  
*Data bus width.*
- bool [useExtVsync](#)  
*In CCIR656 progressive mode, set true to use external VSYNC signal, set false to use internal VSYNC signal decoded from SOF.*

#### 12.4.1.0.0.4 Field Documentation

##### 12.4.1.0.0.4.1 uint16\_t csi\_config\_t::width

##### 12.4.1.0.0.4.2 uint16\_t csi\_config\_t::height

##### 12.4.1.0.0.4.3 uint32\_t csi\_config\_t::polarityFlags

##### 12.4.1.0.0.4.4 uint8\_t csi\_config\_t::bytesPerPixel

- 2: Used for RGB565, YUV422, and so on.
- 3: Used for packed RGB888, packed YUV444, and so on.
- 4: Used for XRGB8888, XYUV444, and so on.

## Data Structure Documentation

12.4.1.0.0.4.5 `uint16_t csi_config_t::linePitch_Bytes`

12.4.1.0.0.4.6 `csi_work_mode_t csi_config_t::workMode`

12.4.1.0.0.4.7 `csi_data_bus_t csi_config_t::dataBus`

12.4.1.0.0.4.8 `bool csi_config_t::useExtVsync`

### 12.4.2 `struct _csi_handle`

Please see the user guide for the details of the CSI driver queue mechanism.

#### Data Fields

- `uint32_t frameBufferQueue` [CSI\_DRIVER\_ACTUAL\_QUEUE\_SIZE]  
*Frame buffer queue.*
- `volatile uint8_t queueUserReadIdx`  
*Application gets full-filled frame buffer from this index.*
- `volatile uint8_t queueUserWriteIdx`  
*Application puts empty frame buffer to this index.*
- `volatile uint8_t queueDrvReadIdx`  
*Driver gets empty frame buffer from this index.*
- `volatile uint8_t queueDrvWriteIdx`  
*Driver puts the full-filled frame buffer to this index.*
- `volatile uint8_t activeBufferNum`  
*How many frame buffers are in progress currently.*
- `volatile uint8_t nextBufferIdx`  
*The CSI frame buffer index to use for next frame.*
- `volatile bool transferStarted`  
*User has called `CSI_TransferStart` to start frame receiving.*
- `volatile bool transferOnGoing`  
*CSI is working and receiving incoming frames.*
- `csi_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*CSI callback function parameter.*

### 12.4.2.0.0.5 Field Documentation

12.4.2.0.0.5.1 `uint32_t csi_handle_t::frameBufferQueue[CSI_DRIVER_ACTUAL_QUEUE_SIZE]`

12.4.2.0.0.5.2 `volatile uint8_t csi_handle_t::queueUserReadIdx`

12.4.2.0.0.5.3 `volatile uint8_t csi_handle_t::queueUserWriteIdx`

12.4.2.0.0.5.4 `volatile uint8_t csi_handle_t::queueDrvReadIdx`

12.4.2.0.0.5.5 `volatile uint8_t csi_handle_t::queueDrvWriteIdx`

12.4.2.0.0.5.6 `volatile uint8_t csi_handle_t::activeBufferNum`

12.4.2.0.0.5.7 `volatile uint8_t csi_handle_t::nextBufferIdx`

12.4.2.0.0.5.8 `volatile bool csi_handle_t::transferStarted`

12.4.2.0.0.5.9 `volatile bool csi_handle_t::transferOnGoing`

12.4.2.0.0.5.10 `csi_transfer_callback_t csi_handle_t::callback`

12.4.2.0.0.5.11 `void* csi_handle_t::userData`

## 12.5 Macro Definition Documentation

12.5.1 `#define CSI_DRIVER_QUEUE_SIZE 4U`

## 12.6 Typedef Documentation

12.6.1 `typedef void(* csi_transfer_callback_t)(CSI_Type *base, csi_handle_t *handle, status_t status, void *userData)`

When a new frame is received and saved to the frame buffer queue, the callback is called and the pass the status `kStatus_CSI_FrameDone` to upper layer.

## 12.7 Enumeration Type Documentation

### 12.7.1 `enum _csi_status`

Enumerator

*kStatus\_CSI\_NoEmptyBuffer* No empty frame buffer in queue to load to CSI.

*kStatus\_CSI\_NoFullBuffer* No full frame buffer in queue to read out.

*kStatus\_CSI\_QueueFull* Queue is full, no room to save new empty buffer.

*kStatus\_CSI\_FrameDone* New frame received and saved to queue.

## Enumeration Type Documentation

### 12.7.2 enum csi\_work\_mode\_t

The CCIR656 interlace mode is not supported currently.

Enumerator

*kCSI\_GatedClockMode* HSYNC, VSYNC, and PIXCLK signals are used.

*kCSI\_NonGatedClockMode* VSYNC, and PIXCLK signals are used.

*kCSI\_CCIR656ProgressiveMode* CCIR656 progressive mode.

### 12.7.3 enum csi\_data\_bus\_t

Currently only support 8-bit width.

Enumerator

*kCSI\_DataBus8Bit* 8-bit data bus.

### 12.7.4 enum \_csi\_polarity\_flags

Enumerator

*kCSI\_HsyncActiveLow* HSYNC is active low.

*kCSI\_HsyncActiveHigh* HSYNC is active high.

*kCSI\_DataLatchOnRisingEdge* Pixel data latched at rising edge of pixel clock.

*kCSI\_DataLatchOnFallingEdge* Pixel data latched at falling edge of pixel clock.

*kCSI\_VsyncActiveHigh* VSYNC is active high.

*kCSI\_VsyncActiveLow* VSYNC is active low.

### 12.7.5 enum csi\_fifo\_t

Enumerator

*kCSI\_RxFifo* RXFIFO.

*kCSI\_StatFifo* STAT FIFO.

*kCSI\_AllFifo* Both RXFIFO and STAT FIFO.

### 12.7.6 enum \_csi\_interrupt\_enable

Enumerator

*kCSI\_EndOfFrameInterruptEnable* End of frame interrupt enable.

***kCSI\_ChangeOfFieldInterruptEnable*** Change of field interrupt enable.  
***kCSI\_StatFifoOverrunInterruptEnable*** STAT FIFO overrun interrupt enable.  
***kCSI\_RxFifoOverrunInterruptEnable*** RXFIFO overrun interrupt enable.  
***kCSI\_StatFifoDmaDoneInterruptEnable*** STAT FIFO DMA done interrupt enable.  
***kCSI\_StatFifoFullInterruptEnable*** STAT FIFO full interrupt enable.  
***kCSI\_RxBuffer1DmaDoneInterruptEnable*** RX frame buffer 1 DMA transfer done.  
***kCSI\_RxBuffer0DmaDoneInterruptEnable*** RX frame buffer 0 DMA transfer done.  
***kCSI\_RxFifoFullInterruptEnable*** RXFIFO full interrupt enable.  
***kCSI\_StartOfFrameInterruptEnable*** Start of frame (SOF) interrupt enable.  
***kCSI\_EccErrorInterruptEnable*** ECC error detection interrupt enable.  
***kCSI\_AhbResErrorInterruptEnable*** AHB response Error interrupt enable.  
***kCSI\_BaseAddrChangeErrorInterruptEnable*** The DMA output buffer base address changes before DMA completed.  
***kCSI\_Field0DoneInterruptEnable*** Field 0 done interrupt enable.  
***kCSI\_Field1DoneInterruptEnable*** Field 1 done interrupt enable.

### 12.7.7 enum\_csi\_flags

The following status register flags can be cleared:

- ***kCSI\_EccErrorFlag***
- ***kCSI\_AhbResErrorFlag***
- ***kCSI\_ChangeOfFieldFlag***
- ***kCSI\_StartOfFrameFlag***
- ***kCSI\_EndOfFrameFlag***
- ***kCSI\_RxBuffer1DmaDoneFlag***
- ***kCSI\_RxBuffer0DmaDoneFlag***
- ***kCSI\_StatFifoDmaDoneFlag***
- ***kCSI\_StatFifoOverrunFlag***
- ***kCSI\_RxFifoOverrunFlag***
- ***kCSI\_Field0DoneFlag***
- ***kCSI\_Field1DoneFlag***
- ***kCSI\_BaseAddrChangeErrorFlag***

Enumerator

***kCSI\_RxFifoDataReadyFlag*** RXFIFO data ready.  
***kCSI\_EccErrorFlag*** ECC error detected.  
***kCSI\_AhbResErrorFlag*** Hresponse (AHB bus response) Error.  
***kCSI\_ChangeOfFieldFlag*** Change of field.  
***kCSI\_Field0PresentFlag*** Field 0 present in CCIR mode.  
***kCSI\_Field1PresentFlag*** Field 1 present in CCIR mode.  
***kCSI\_StartOfFrameFlag*** Start of frame (SOF) detected.  
***kCSI\_EndOfFrameFlag*** End of frame (EOF) detected.

## Function Documentation

*kCSI\_RxFifoFullFlag* RXFIFO full (Number of data reaches trigger level).  
*kCSI\_RxBuffer1DmaDoneFlag* RX frame buffer 1 DMA transfer done.  
*kCSI\_RxBuffer0DmaDoneFlag* RX frame buffer 0 DMA transfer done.  
*kCSI\_StatFifoFullFlag* STAT FIFO full (Reach trigger level).  
*kCSI\_StatFifoDmaDoneFlag* STAT FIFO DMA transfer done.  
*kCSI\_StatFifoOverrunFlag* STAT FIFO overrun.  
*kCSI\_RxFifoOverrunFlag* RXFIFO overrun.  
*kCSI\_Field0DoneFlag* Field 0 transfer done.  
*kCSI\_Field1DoneFlag* Field 1 transfer done.  
*kCSI\_BaseAddrChangeErrorFlag* The DMA output buffer base address changes before DMA completed.

## 12.8 Function Documentation

### 12.8.1 `status_t CSI_Init ( CSI_Type * base, const csi_config_t * config )`

This function enables the CSI peripheral clock, and resets the CSI registers.

Parameters

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

Return values

<i>kStatus_Success</i>	Initialize successfully.
<i>kStatus_InvalidArgument</i>	Initialize failed because of invalid argument.

### 12.8.2 `void CSI_Deinit ( CSI_Type * base )`

This function disables the CSI peripheral clock.

Parameters

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

### 12.8.3 `void CSI_Reset ( CSI_Type * base )`

This function resets the CSI peripheral registers to default status.



## Parameters

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

**12.8.4 void CSI\_GetDefaultConfig ( csi\_config\_t \* config )**

The default configuration value is:

```
config->width = 320U;
config->height = 240U;
config->polarityFlags = kCSI_HsyncActiveHigh |
    kCSI_DataLatchOnRisingEdge;
config->bytesPerPixel = 2U;
config->linePitch_Bytes = 320U * 2U;
config->workMode = kCSI_GatedClockMode;
config->dataBus = kCSI_DataBus8Bit;
config->useExtVsync = true;
```

## Parameters

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

**12.8.5 void CSI\_ClearFifo ( CSI\_Type \* base, csi\_fifo\_t fifo )**

This function clears the CSI FIFO.

## Parameters

<i>base</i>	CSI peripheral base address.
<i>fifo</i>	The FIFO to clear.

**12.8.6 void CSI\_ReflashFifoDma ( CSI\_Type \* base, csi\_fifo\_t fifo )**

This function reflashes the CSI FIFO DMA.

For RXFIFO, there are two frame buffers. When the CSI module started, it saves the frames to frame buffer 0 then frame buffer 1, the two buffers will be written by turns. After reflash DMA using this function, the CSI is reset to save frame to buffer 0.

## Function Documentation

Parameters

<i>base</i>	CSI peripheral base address.
<i>fifo</i>	The FIFO DMA to reflash.

**12.8.7 void CSI\_EnableFifoDmaRequest ( CSI\_Type \* *base*, csi\_fifo\_t *fifo*, bool *enable* )**

Parameters

<i>base</i>	CSI peripheral base address.
<i>fifo</i>	The FIFO DMA reques to enable or disable.
<i>enable</i>	True to enable, false to disable.

**12.8.8 static void CSI\_Start ( CSI\_Type \* *base* ) [inline], [static]**

Parameters

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

**12.8.9 static void CSI\_Stop ( CSI\_Type \* *base* ) [inline], [static]**

Parameters

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

**12.8.10 void CSI\_SetRxBufferAddr ( CSI\_Type \* *base*, uint8\_t *index*, uint32\_t *addr* )**

Parameters

<i>base</i>	CSI peripheral base address.
<i>index</i>	Buffer index.
<i>addr</i>	Frame buffer address to set.

### 12.8.11 void CSI\_EnableInterrupts ( CSI\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	CSI peripheral base address.
<i>mask</i>	The interrupts to enable, pass in as OR'ed value of <a href="#">_csi_interrupt_enable</a> .

### 12.8.12 void CSI\_DisableInterrupts ( CSI\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	CSI peripheral base address.
<i>mask</i>	The interrupts to disable, pass in as OR'ed value of <a href="#">_csi_interrupt_enable</a> .

### 12.8.13 static uint32\_t CSI\_GetStatusFlags ( CSI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

status flag, it is OR'ed value of [\\_csi\\_flags](#).

### 12.8.14 static void CSI\_ClearStatusFlags ( CSI\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The flags to clear are passed in as OR'ed value of [\\_csi\\_flags](#). The following flags are cleared automatically by hardware:

- [kCSI\\_RxFifoFullFlag](#),

## Function Documentation

- [kCSI\\_StatFifoFullFlag](#),
- [kCSI\\_Field0PresentFlag](#),
- [kCSI\\_Field1PresentFlag](#),
- [kCSI\\_RxFifoDataReadyFlag](#),

Parameters

<i>base</i>	CSI peripheral base address.
<i>statusMask</i>	The status flags mask, OR'ed value of <a href="#">_csi_flags</a> .

### 12.8.15 `status_t CSI_TransferCreateHandle ( CSI_Type * base, csi_handle_t * handle, csi_transfer_callback_t callback, void * userData )`

This function initializes CSI handle, it should be called before any other CSI transactional functions.

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.
<i>callback</i>	Callback function for CSI transfer.
<i>userData</i>	Callback function parameter.

Return values

<i>kStatus_Success</i>	Handle created successfully.
------------------------	------------------------------

### 12.8.16 `status_t CSI_TransferStart ( CSI_Type * base, csi_handle_t * handle )`

When the empty frame buffers have been submit to CSI driver using function [CSI\\_TransferSubmitEmpty-Buffer](#), user could call this function to start the transfer. The incoming frame will be saved to the empty frame buffer, and user could be optionally notified through callback function.

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.

Return values

<i>kStatus_Success</i>	Started successfully.
<i>kStatus_CSI_NoEmpty-Buffer</i>	Could not start because no empty frame buffer in queue.

### 12.8.17 **status\_t CSI\_TransferStop ( CSI\_Type \* *base*, csi\_handle\_t \* *handle* )**

The driver does not clean the full frame buffers in queue. In other words, after calling this function, user still could get the full frame buffers in queue using function [CSI\\_TransferGetFullBuffer](#).

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.

Return values

<i>kStatus_Success</i>	Stoped successfully.
------------------------	----------------------

### 12.8.18 **status\_t CSI\_TransferSubmitEmptyBuffer ( CSI\_Type \* *base*, csi\_handle\_t \* *handle*, uint32\_t *frameBuffer* )**

This function could be called before [CSI\\_TransferStart](#) or after [CSI\\_TransferStart](#). If there is no room in queue to store the empty frame buffer, this function returns error.

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.
<i>frameBuffer</i>	Empty frame buffer to submit.

Return values

<i>kStatus_Success</i>	Started successfully.
------------------------	-----------------------

## Function Documentation

<i>kStatus_CSI_QueueFull</i>	Could not submit because there is no room in queue.
------------------------------	-----------------------------------------------------

### 12.8.19 **status\_t CSI\_TransferGetFullBuffer ( CSI\_Type \* *base*, csi\_handle\_t \* *handle*, uint32\_t \* *frameBuffer* )**

After the transfer started using function [CSI\\_TransferStart](#), the incoming frames will be saved to the empty frame buffers in queue. This function gets the full-filled frame buffer from the queue. If there is no full frame buffer in queue, this function returns error.

#### Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.
<i>frameBuffer</i>	Full frame buffer.

#### Return values

<i>kStatus_Success</i>	Started successfully.
<i>kStatus_CSI_NoFull-Buffer</i>	There is no full frame buffer in queue.

### 12.8.20 **void CSI\_TransferHandleIRQ ( CSI\_Type \* *base*, csi\_handle\_t \* *handle* )**

This function handles the CSI IRQ request to work with CSI driver transactional APIs.

#### Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	CSI handle pointer.

## Chapter 13

# DCDC: DCDC Converter

### 13.1 Overview

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

This part describes the programming interface of the DCDC Converter Peripheral driver. The DCDC converter module is a synchronous buck mode DCDC converter. It can produce single outputs for SoC peripherals and external devices with high conversion efficiency. The converter can be operated in continuous or pulsed mode.

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

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

### 13.2 Function groups

#### 13.2.1 Initialization and deinitialization

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

#### 13.2.2 Status

Provides functions to get the DCDC status.

#### 13.2.3 Misc control

Provides functions to set the DCDC's miscellaneous control.

## Application guideline

### 13.3 Application guideline

#### 13.3.1 Continuous conduction mode

#### 13.3.2 Discontinuous conduction mode

### Data Structures

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

### Macros

- #define `FSL_DCDC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*DCDC driver version.*

### Enumerations

- enum `_dcdc_status_flags_t` { `kDCDC_LockedOKStatus` = (1U << 0U) }  
*DCDC status flags.*
- enum `dcdc_comparator_current_bias_t` {  
`kDCDC_ComparatorCurrentBias50nA` = 0U,  
`kDCDC_ComparatorCurrentBias100nA` = 1U,  
`kDCDC_ComparatorCurrentBias200nA` = 2U,  
`kDCDC_ComparatorCurrentBias400nA` = 3U }  
*The current bias of low power comparator.*
- enum `dcdc_over_current_threshold_t` {  
`kDCDC_OverCurrentThresholdAlt0` = 0U,  
`kDCDC_OverCurrentThresholdAlt1` = 1U,  
`kDCDC_OverCurrentThresholdAlt2` = 2U,  
`kDCDC_OverCurrentThresholdAlt3` = 3U }  
*The threshold of over current detection.*
- enum `dcdc_peak_current_threshold_t` {  
`kDCDC_PeakCurrentThresholdAlt0` = 0U,  
`kDCDC_PeakCurrentThresholdAlt1` = 1U,  
`kDCDC_PeakCurrentThresholdAlt2` = 2U,  
`kDCDC_PeakCurrentThresholdAlt3` = 3U,  
`kDCDC_PeakCurrentThresholdAlt4` = 4U,  
`kDCDC_PeakCurrentThresholdAlt5` = 5U }  
*The threshold if peak current detection.*



- enum `dcdc_count_charging_time_period_t` {  
`kDCDC_CountChargingTimePeriod8Cycle = 0U`,  
`kDCDC_CountChargingTimePeriod16Cycle = 1U` }  
*The period of counting the charging times in power save mode.*
- enum `dcdc_count_charging_time_threshold_t` {  
`kDCDC_CountChargingTimeThreshold32 = 0U`,  
`kDCDC_CountChargingTimeThreshold64 = 1U`,  
`kDCDC_CountChargingTimeThreshold16 = 2U`,  
`kDCDC_CountChargingTimeThreshold8 = 3U` }  
*The threshold of the counting number of charging times.*
- enum `dcdc_clock_source_t` {  
`kDCDC_ClockAutoSwitch = 0U`,  
`kDCDC_ClockInternalOsc = 1U`,  
`kDCDC_ClockExternalOsc = 2U` }  
*Oscillator clock option.*

## Variables

- bool `dcdc_detection_config_t::enableXtalokDetection`  
*Enable xtalok detection circuit.*
- bool `dcdc_detection_config_t::powerDownOverVoltageDetection`  
*Power down over-voltage detection comparator.*
- bool `dcdc_detection_config_t::powerDownLowVoltageDetection`  
*Power down low-voltage detection comparator.*
- bool `dcdc_detection_config_t::powerDownOverCurrentDetection`  
*Power down over-current detection.*
- bool `dcdc_detection_config_t::powerDownPeakCurrentDetection`  
*Power down peak-current detection.*
- bool `dcdc_detection_config_t::powerDownZeroCrossDetection`  
*Power down the zero cross detection function for discontinuous conductor mode.*
- `dcdc_over_current_threshold_t dcdc_detection_config_t::OverCurrentThreshold`  
*The threshold of over current detection.*
- `dcdc_peak_current_threshold_t dcdc_detection_config_t::PeakCurrentThreshold`  
*The threshold of peak current detection.*
- bool `dcdc_loop_control_config_t::enableCommonHysteresis`  
*Enable hysteresis in switching converter common mode analog comparators.*
- bool `dcdc_loop_control_config_t::enableCommonThresholdDetection`  
*Increase the threshold detection for common mode analog comparator.*
- bool `dcdc_loop_control_config_t::enableInvertHysteresisSign`  
*Invert the sign of the hysteresis in DC-DC analog comparators.*
- bool `dcdc_loop_control_config_t::enableRCThresholdDetection`  
*Increase the threshold detection for RC scale circuit.*
- `uint32_t dcdc_loop_control_config_t::enableRCScaleCircuit`  
*Available range is 0~7.*
- `uint32_t dcdc_loop_control_config_t::complementFeedForwardStep`  
*Available range is 0~7.*
- `uint32_t dcdc_loop_control_config_t::controlParameterMagnitude`  
*Available range is 0~15.*
- `uint32_t dcdc_loop_control_config_t::integralProportionalRatio`  
*Available range is 0~3. Ratio of integral control parameter to proportional control parameter in the switch-*

## Application guideline

- ing DC-DC converter, and can be used to optimize efficiency and loop response.
- bool [dcdc\\_low\\_power\\_config\\_t::enableOverloadDetection](#)  
*Enable the overload detection in power save mode, if current is larger than the overloading threshold (typical value is 50 mA), DCDC will switch to the run mode automatically.*
- bool [dcdc\\_low\\_power\\_config\\_t::enableAdjustHystereticValue](#)  
*Adjust hysteretic value in low power from 12.5mV to 25mV.*
- [dcdc\\_count\\_charging\\_time\\_period\\_t dcdc\\_low\\_power\\_config\\_t::countChargingTimePeriod](#)  
*The period of counting the charging times in power save mode.*
- [dcdc\\_count\\_charging\\_time\\_threshold\\_t dcdc\\_low\\_power\\_config\\_t::countChargingTimeThreshold](#)  
*the threshold of the counting number of charging times during the period that `lp_overload_freq_sel` sets in power save mode.*
- bool [dcdc\\_internal\\_regulator\\_config\\_t::enableLoadResistor](#)  
*control the load resistor of the internal regulator of DCDC, the load resistor is connected as default "true", and need set to "false" to disconnect the load resistor.*
- [uint32\\_t dcdc\\_internal\\_regulator\\_config\\_t::feedbackPoint](#)  
*Available range is 0~3.*
- bool [dcdc\\_min\\_power\\_config\\_t::enableUseHalfFreqForContinuous](#)  
*Set DCDC clock to half frequency for the continuous mode.*

## Initialization and deinitialization

- void [DCDC\\_Init](#) (DCDC\_Type \*base)  
*Enable the access to DCDC registers.*
- void [DCDC\\_Deinit](#) (DCDC\_Type \*base)  
*Disable the access to DCDC registers.*

## Status

- [uint32\\_t DCDC\\_GetstatusFlags](#) (DCDC\_Type \*base)  
*Get DCDC status flags.*

## Misc control.

- static void [DCDC\\_EnableOutputRangeComparator](#) (DCDC\_Type \*base, bool enable)  
*Enable the output range comparator.*
- void [DCDC\\_SetClockSource](#) (DCDC\_Type \*base, [dcdc\\_clock\\_source\\_t](#) clockSource)  
*Configure the DCDC clock source.*
- void [DCDC\\_GetDefaultDetectionConfig](#) ([dcdc\\_detection\\_config\\_t](#) \*config)  
*Get the default setting for detection configuration.*
- void [DCDC\\_SetDetectionConfig](#) (DCDC\_Type \*base, const [dcdc\\_detection\\_config\\_t](#) \*config)  
*Configure the DCDC detection.*
- void [DCDC\\_GetDefaultLowPowerConfig](#) ([dcdc\\_low\\_power\\_config\\_t](#) \*config)  
*Get the default setting for low power configuration.*
- void [DCDC\\_SetLowPowerConfig](#) (DCDC\_Type \*base, const [dcdc\\_low\\_power\\_config\\_t](#) \*config)  
*Configure the DCDC low power.*
- void [DCDC\\_ResetCurrentAlertSignal](#) (DCDC\_Type \*base, bool enable)  
*Reset current alert signal.*
- static void [DCDC\\_SetBandgapVoltageTrimValue](#) (DCDC\_Type \*base, [uint32\\_t](#) trimValue)  
*Set the bangap trim value to trim bandgap voltage.*
- void [DCDC\\_GetDefaultLoopControlConfig](#) ([dcdc\\_loop\\_control\\_config\\_t](#) \*config)  
*Get the default setting for loop control configuration.*

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

## Application guideline.

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

## 13.4 Data Structure Documentation

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

#### Data Fields

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

## Data Structure Documentation

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

#### Data Fields

- bool [enableCommonHysteresis](#)  
*Enable hysteresis in switching converter common mode analog comparators.*
- bool [enableCommonThresholdDetection](#)  
*Increase the threshold detection for common mode analog comparator.*
- bool [enableInvertHysteresisSign](#)  
*Invert the sign of the hysteresis in DC-DC analog comparators.*
- bool [enableRCThresholdDetection](#)  
*Increase the threshold detection for RC scale circuit.*
- uint32\_t [enableRCScaleCircuit](#)  
*Available range is 0~7.*
- uint32\_t [complementFeedForwardStep](#)  
*Available range is 0~7.*
- uint32\_t [controlParameterMagnitude](#)  
*Available range is 0~15.*
- uint32\_t [integralProportionalRatio](#)  
*Available range is 0~3. Ratio of integral control parameter to proportional control parameter in the switching DC-DC converter, and can be used to optimize efficiency and loop response.*

### 13.4.3 struct dcdc\_low\_power\_config\_t

#### Data Fields

- bool [enableOverloadDetection](#)  
*Enable the overload detection in power save mode, if current is larger than the overloading threshold (typical value is 50 mA), DCDC will switch to the run mode automatically.*
- bool [enableAdjustHystereticValue](#)  
*Adjust hysteretic value in low power from 12.5mV to 25mV.*
- [dcdc\\_count\\_charging\\_time\\_period\\_t](#) [countChargingTimePeriod](#)  
*The period of counting the charging times in power save mode.*
- [dcdc\\_count\\_charging\\_time\\_threshold\\_t](#) [countChargingTimeThreshold](#)  
*the threshold of the counting number of charging times during the period that lp\_overload\_freq\_sel sets in power save mode.*

### 13.4.4 struct dcdc\_internal\_regulator\_config\_t

#### Data Fields

- bool [enableLoadResistor](#)  
*control the load resistor of the internal regulator of DCDC, the load resistor is connected as default "true", and need set to "false" to disconnect the load resistor.*
- uint32\_t [feedbackPoint](#)  
*Available range is 0~3.*

### 13.4.5 struct dcdc\_min\_power\_config\_t

#### Data Fields

- bool [enableUseHalfFreqForContinuous](#)  
Set DCDC clock to half frequency for the continuous mode.

## 13.5 Macro Definition Documentation

### 13.5.1 #define FSL\_DCDC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

## 13.6 Enumeration Type Documentation

### 13.6.1 enum \_dcdc\_status\_flags\_t

Enumerator

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

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

Enumerator

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

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

Enumerator

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

## Function Documentation

### 13.6.4 enum dc\_dc\_peak\_current\_threshold\_t

Enumerator

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

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

Enumerator

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

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

Enumerator

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

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

Enumerator

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

## 13.7 Function Documentation

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

Parameters

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

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

Parameters

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

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

Parameters

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

Returns

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

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

The output range comparator is disabled by default.

Parameters

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

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

## Function Documentation

### Parameters

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

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

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

```
* config->enableXtalokDetection = false;
* config->powerDownOverVoltageDetection = true;
* config->powerDownLowVoltageDetection = false;
* config->powerDownOverCurrentDetection = true;
* config->powerDownPeakCurrentDetection = true;
* config->powerDownZeroCrossDetection = true;
* config->OverCurrentThreshold = kDCDC_OverCurrentThresholdAlt0;
* config->PeakCurrentThreshold = kDCDC_PeakCurrentThresholdAlt0;
*
```

### Parameters

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

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

### Parameters

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

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

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

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



Parameters

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

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

Parameters

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

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

Alert signal is generate by peak current detection.

Parameters

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

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

Parameters

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

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

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

```
* config->enableCommonHysteresis = false;
* config->enableCommonThresholdDetection = false;
* config->enableInvertHysteresisSign = false;
* config->enableRCThresholdDetection = false;
```

## Function Documentation

```
* config->enableRCScaleCircuit = 0U;  
* config->complementFeedForwardStep = 0U;  
* config->controlParameterMagnitude = 2U;  
* config->integralProportionalRatio = 2U;  
*
```

### Parameters

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

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

### Parameters

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

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

### Parameters

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

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

### Parameters

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

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

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

## Function Documentation

Parameters

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

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

pwd\_zcd=0x0; pwd\_cmp\_offset=0x0; dcdc\_loopctrl\_en\_rcscale=0x3 or 0x5; DCM\_set\_ctrl=1'b1;

Parameters

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

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

pwd\_zcd=0x1; pwd\_cmp\_offset=0x0; dcdc\_loopctrl\_en\_rcscale=0x3;

Parameters

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

## 13.8 Variable Documentation

**13.8.1 bool dcdc\_detection\_config\_t::enableXtalokDetection**

**13.8.2 bool dcdc\_detection\_config\_t::powerDownOverVoltageDetection**

**13.8.3 bool dcdc\_detection\_config\_t::powerDownLowVoltageDetection**

**13.8.4 bool dcdc\_detection\_config\_t::powerDownOverCurrentDetection**

**13.8.5 bool dcdc\_detection\_config\_t::powerDownPeakCurrentDetection**

**13.8.6 bool dcdc\_detection\_config\_t::powerDownZeroCrossDetection**

**13.8.7 dcdc\_over\_current\_threshold\_t dcdc\_detection\_config\_t::OverCurrent-Threshold**

**13.8.8 dcdc\_peak\_current\_threshold\_t dcdc\_detection\_config\_t::PeakCurrent-Threshold**

**13.8.9 bool dcdc\_loop\_control\_config\_t::enableCommonHysteresis**

This feature will improve transient supply ripple and efficiency.

**13.8.10 bool dcdc\_loop\_control\_config\_t::enableCommonThresholdDetection**

**13.8.11 bool dcdc\_loop\_control\_config\_t::enableInvertHysteresisSign**

**13.8.12 bool dcdc\_loop\_control\_config\_t::enableRCThresholdDetection**

**13.8.13 uint32\_t dcdc\_loop\_control\_config\_t::enableRCScaleCircuit**

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

**13.8.14 uint32\_t dcdc\_loop\_control\_config\_t::complementFeedForwardStep**

Two's complement feed forward step in duty cycle in the switching DC-DC converter. Each time this field makes a transition from 0x0, the loop filter of the DC-DC converter is stepped once by a value proportional

## Variable Documentation

to the change. This can be used to force a certain control loop behavior, such as improving response under known heavy load transients.

### 13.8.15 `uint32_t dcdc_loop_control_config_t::controlParameterMagnitude`

Magnitude of proportional control parameter in the switching DC-DC converter control loop.

### 13.8.16 `uint32_t dcdc_loop_control_config_t::integralProportionalRatio`

### 13.8.17 `bool dcdc_low_power_config_t::enableOverloadDetection`

### 13.8.18 `bool dcdc_low_power_config_t::enableAdjustHystereticValue`

### 13.8.19 `dcdc_count_charging_time_period_t dcdc_low_power_config_t::count-ChargingTimePeriod`

### 13.8.20 `dcdc_count_charging_time_threshold_t dcdc_low_power_config_t::count-ChargingTimeThreshold`

### 13.8.21 `bool dcdc_internal_regulator_config_t::enableLoadResistor`

### 13.8.22 `uint32_t dcdc_internal_regulator_config_t::feedbackPoint`

Select the feedback point of the internal regulator.

### 13.8.23 `bool dcdc_min_power_config_t::enableUseHalfFreqForContinuous`

## Chapter 14

### DCP: Data Co-Processor

#### 14.1 Overview

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

The driver comprises two sets of API functions.

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

The DCP work packets (descriptors) are placed on the system stack during the blocking API calls. The driver uses critical part (implemented as global interrupt enable/disable) for a short time whenever it needs to pass DCP work packets to DCP channel for processing. Therefore, the driver functions are designed to be re-entrant and as a consequence, one CPU thread can call one blocking API, such as AES Encrypt, while other CPU thread can call another blocking API, such as SHA-256 Update. The blocking functions provide typical interface to upper layer or application software. In the second set, non-blocking variants of the first set APIs are provided. Internally, the blocking APIs are implemented as a non-blocking operation start, followed by a blocking wait (CPU polling DCP work packet's status word) for an operation completion. The non-blocking functions allow upper layer to inject an application specific operation after the DCP operation start and DCP channel complete events. RTOS event wait and RTOS event set can be an example of such an operation.

#### 14.2 DCP Driver Initialization and Configuration

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

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

#### Key Management

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

## DCP Driver Examples

### 14.3 Comments about API usage in RTOS

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

### 14.4 Comments about API usage in interrupt handler

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

### 14.5 DCP Driver Examples

#### 14.5.1 Simple examples

Encrypt plaintext by AES engine Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/DCP`

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

Compute hash (SHA-256) Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/DCP`

## Modules

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

## Data Structures

- struct [dcp\\_work\\_packet\\_t](#)  
*DCP's work packet. [More...](#)*
- struct [dcp\\_handle\\_t](#)  
*Specify DCP's key resource and DCP channel. [More...](#)*
- struct [dcp\\_context\\_t](#)  
*DCP's context buffer, used by DCP for context switching between channels. [More...](#)*
- struct [dcp\\_config\\_t](#)  
*DCP's configuration structure. [More...](#)*



## Enumerations

- enum `_dcp_ch_enable_t` {  
`kDCP_chDisable` = 0U,  
`kDCP_ch0Enable` = 1U,  
`kDCP_ch1Enable` = 2U,  
`kDCP_ch2Enable` = 4U,  
`kDCP_ch3Enable` = 8U,  
`kDCP_chEnableAll` = 15U }  
*DCP channel enable.*
- enum `_dcp_ch_int_enable_t` {  
`kDCP_chIntDisable` = 0U,  
`kDCP_ch0IntEnable` = 1U,  
`kDCP_ch1IntEnable` = 2U,  
`kDCP_ch2IntEnable` = 4U,  
`kDCP_ch3IntEnable` = 8U }  
*DCP interrupt enable.*
- enum `dcp_channel_t` {  
`kDCP_Channel0` = (1u << 16),  
`kDCP_Channel1` = (1u << 17),  
`kDCP_Channel2` = (1u << 18),  
`kDCP_Channel3` = (1u << 19) }  
*DCP channel selection.*
- enum `dcp_key_slot_t` {  
`kDCP_KeySlot0` = 0U,  
`kDCP_KeySlot1` = 1U,  
`kDCP_KeySlot2` = 2U,  
`kDCP_KeySlot3` = 3U,  
`kDCP_OtpKey` = 4U,  
`kDCP_OtpUniqueKey` = 5U,  
`kDCP_PayloadKey` = 6U }  
*DCP key slot selection.*

## Functions

- void `DCP_Init` (DCP\_Type \*base, const `dcp_config_t` \*config)  
*Enables clock to and enables DCP.*
- void `DCP_Deinit` (DCP\_Type \*base)  
*Disable DCP clock.*
- void `DCP_GetDefaultConfig` (`dcp_config_t` \*config)  
*Gets the default configuration structure.*
- status\_t `DCP_WaitForChannelComplete` (DCP\_Type \*base, `dcp_handle_t` \*handle)  
*Poll and wait on DCP channel.*

## Driver version

- #define `FSL_DCP_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*DCP driver version.*

## Data Structure Documentation

### 14.6 Data Structure Documentation

#### 14.6.1 struct dcp\_work\_packet\_t

#### 14.6.2 struct dcp\_handle\_t

##### Data Fields

- [dcp\\_channel\\_t channel](#)  
*Specify DCP channel.*
- [dcp\\_key\\_slot\\_t keySlot](#)  
*For operations with key (such as AES encryption/decryption), specify DCP key slot.*

##### 14.6.2.0.0.6 Field Documentation

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

14.6.2.0.0.6.2 [dcp\\_key\\_slot\\_t dcp\\_handle\\_t::keySlot](#)

#### 14.6.3 struct dcp\_context\_t

#### 14.6.4 struct dcp\_config\_t

##### Data Fields

- bool [gatherResidualWrites](#)  
*Enable the ragged writes to the unaligned buffers.*
- bool [enableContextCaching](#)  
*Enable the caching of contexts between the operations.*
- bool [enableContextSwitching](#)  
*Enable automatic context switching for the channels.*
- uint8\_t [enableChannel](#)  
*DCP channel enable.*
- uint8\_t [enableChannelInterrupt](#)  
*Per-channel interrupt enable.*

##### 14.6.4.0.0.7 Field Documentation

14.6.4.0.0.7.1 [bool dcp\\_config\\_t::gatherResidualWrites](#)

14.6.4.0.0.7.2 [bool dcp\\_config\\_t::enableContextCaching](#)

14.6.4.0.0.7.3 [bool dcp\\_config\\_t::enableContextSwitching](#)

14.6.4.0.0.7.4 [uint8\\_t dcp\\_config\\_t::enableChannel](#)

14.6.4.0.0.7.5 [uint8\\_t dcp\\_config\\_t::enableChannelInterrupt](#)

## 14.7 Macro Definition Documentation

### 14.7.1 #define FSL\_DCP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

Current version: 2.0.0

Change log:

- Version 2.0.0
  - Initial version

## 14.8 Enumeration Type Documentation

### 14.8.1 enum \_dcp\_ch\_enable\_t

Enumerator

*kDCP\_chDisable* DCP channel disable.  
*kDCP\_ch0Enable* DCP channel 0 enable.  
*kDCP\_ch1Enable* DCP channel 1 enable.  
*kDCP\_ch2Enable* DCP channel 2 enable.  
*kDCP\_ch3Enable* DCP channel 3 enable.  
*kDCP\_chEnableAll* DCP channel enable all.

### 14.8.2 enum \_dcp\_ch\_int\_enable\_t

Enumerator

*kDCP\_chIntDisable* DCP interrupts disable.  
*kDCP\_ch0IntEnable* DCP channel 0 interrupt enable.  
*kDCP\_ch1IntEnable* DCP channel 1 interrupt enable.  
*kDCP\_ch2IntEnable* DCP channel 2 interrupt enable.  
*kDCP\_ch3IntEnable* DCP channel 3 interrupt enable.

### 14.8.3 enum dcp\_channel\_t

Enumerator

*kDCP\_Channel0* DCP channel 0.  
*kDCP\_Channel1* DCP channel 1.  
*kDCP\_Channel2* DCP channel 2.  
*kDCP\_Channel3* DCP channel 3.

## Function Documentation

### 14.8.4 enum dcp\_key\_slot\_t

Enumerator

- kDCP\_KeySlot0* DCP key slot 0.
- kDCP\_KeySlot1* DCP key slot 1.
- kDCP\_KeySlot2* DCP key slot 2.
- kDCP\_KeySlot3* DCP key slot 3.
- kDCP\_OtpKey* DCP OTP key.
- kDCP\_OtpUniqueKey* DCP unique OTP key.
- kDCP\_PayloadKey* DCP payload key.

## 14.9 Function Documentation

### 14.9.1 void DCP\_Init ( DCP\_Type \* *base*, const dcp\_config\_t \* *config* )

Enable DCP clock and configure DCP.

Parameters

<i>base</i>	DCP base address
<i>config</i>	Pointer to configuration structure.

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

Reset DCP and Disable DCP clock.

Parameters

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

### 14.9.3 void DCP\_GetDefaultConfig ( dcp\_config\_t \* *config* )

This function initializes the DCP configuration structure to a default value. The default values are as follows. `dcpConfig->gatherResidualWrites = true;` `dcpConfig->enableContextCaching = true;` `dcpConfig->enableContextSwitching = true;` `dcpConfig->enableChannel = kDCP_chEnableAll;` `dcpConfig->enableChannelInterrupt = kDCP_chIntDisable;`

Parameters

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

#### 14.9.4 **status\_t DCP\_WaitForChannelComplete ( DCP\_Type \* *base*, dcp\_handle\_t \* *handle* )**

Polls the specified DCP channel until current it completes activity.

Parameters

<i>base</i>	DCP peripheral base address.
<i>handle</i>	Specifies DCP channel.

Returns

kStatus\_Success When data processing completes without error.

kStatus\_Fail When error occurs.

## DCP AES blocking driver

### 14.10 DCP AES blocking driver

#### 14.10.1 Overview

This section describes the programming interface of the DCP AES blocking driver.

#### Macros

- #define `DCP_AES_BLOCK_SIZE` 16  
*AES block size in bytes.*

#### Functions

- status\_t `DCP_AES_SetKey` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*key, size\_t keySize)  
*Set AES key to `dcp_handle_t` struct and optionally to DCP.*
- status\_t `DCP_AES_EncryptEcb` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size)  
*Encrypts AES on one or multiple 128-bit block(s).*
- status\_t `DCP_AES_DecryptEcb` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size)  
*Decrypts AES on one or multiple 128-bit block(s).*
- status\_t `DCP_AES_EncryptCbc` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t iv[16])  
*Encrypts AES using CBC block mode.*
- status\_t `DCP_AES_DecryptCbc` (DCP\_Type \*base, `dcp_handle_t` \*handle, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t iv[16])  
*Decrypts AES using CBC block mode.*

#### 14.10.2 Function Documentation

##### 14.10.2.1 status\_t `DCP_AES_SetKey` ( DCP\_Type \* base, dcp\_handle\_t \* handle, const uint8\_t \* key, size\_t keySize )

Sets the AES key for encryption/decryption with the `dcp_handle_t` structure. The `dcp_handle_t` input argument specifies keySlot. If the keySlot is `kDCP_OtpKey`, the function will check the `OTP_KEY_READY` bit and will return it's ready to use status. For other keySlot selections, the function will copy and hold the key in `dcp_handle_t` struct. If the keySlot is one of the four DCP SRAM-based keys (one of `kDCP_KeySlot0`, `kDCP_KeySlot1`, `kDCP_KeySlot2`, `kDCP_KeySlot3`), this function will also load the supplied key to the specified keySlot in DCP.

## Parameters

<i>base</i>	DCP peripheral base address.
<i>handle</i>	Handle used for the request.
<i>key</i>	0-mod-4 aligned pointer to AES key.
<i>keySize</i>	AES key size in bytes. Shall equal 16.

## Returns

status from set key operation

**14.10.2.2** `status_t DCP_AES_EncryptEcb ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size )`

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

## Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.

## Returns

Status from encrypt operation

**14.10.2.3** `status_t DCP_AES_DecryptEcb ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size )`

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

## Parameters

---

## DCP AES blocking driver

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request.
	<i>ciphertext</i>	Input plain text to encrypt
out	<i>plaintext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.

### Returns

Status from decrypt operation

**14.10.2.4** `status_t DCP_AES_EncryptCbc ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[16] )`

Encrypts AES using CBC block mode. The source plaintext and destination ciphertext can overlap in system memory.

### Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.

### Returns

Status from encrypt operation

**14.10.2.5** `status_t DCP_AES_DecryptCbc ( DCP_Type * base, dcp_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[16] )`

Decrypts AES using CBC block mode. The source ciphertext and destination plaintext can overlap in system memory.



## Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.

## Returns

Status from decrypt operation

## DCP AES non-blocking driver

### 14.11 DCP AES non-blocking driver

#### 14.11.1 Overview

This section describes the programming interface of the DCP AES non-blocking driver.

#### Functions

- status\_t [DCP\\_AES\\_EncryptEcbNonBlocking](#) (DCP\_Type \*base, [dcp\\_handle\\_t](#) \*handle, [dcp\\_work\\_packet\\_t](#) \*dcpPacket, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size)  
*Encrypts AES using the ECB block mode.*
- status\_t [DCP\\_AES\\_DecryptEcbNonBlocking](#) (DCP\_Type \*base, [dcp\\_handle\\_t](#) \*handle, [dcp\\_work\\_packet\\_t](#) \*dcpPacket, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size)  
*Decrypts AES using ECB block mode.*
- status\_t [DCP\\_AES\\_EncryptCbcNonBlocking](#) (DCP\_Type \*base, [dcp\\_handle\\_t](#) \*handle, [dcp\\_work\\_packet\\_t](#) \*dcpPacket, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t \*iv)  
*Encrypts AES using CBC block mode.*
- status\_t [DCP\\_AES\\_DecryptCbcNonBlocking](#) (DCP\_Type \*base, [dcp\\_handle\\_t](#) \*handle, [dcp\\_work\\_packet\\_t](#) \*dcpPacket, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t \*iv)  
*Decrypts AES using CBC block mode.*

#### 14.11.2 Function Documentation

##### 14.11.2.1 status\_t [DCP\\_AES\\_EncryptEcbNonBlocking](#) ( DCP\_Type \* *base*, [dcp\\_handle\\_t](#) \* *handle*, [dcp\\_work\\_packet\\_t](#) \* *dcpPacket*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, size\_t *size* )

Puts AES ECB encrypt work packet to DCP channel.

Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request.
out	<i>dcpPacket</i>	Memory for the DCP work packet.
	<i>plaintext</i>	Input plain text to encrypt.
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

- kStatus\_Success The work packet has been scheduled at DCP channel.
- kStatus\_DCP\_Again The DCP channel is busy processing previous request.

**14.11.2.2** `status_t DCP_AES_DecryptEcbNonBlocking ( DCP_Type * base, dcp_handle_t * handle, dcp_work_packet_t * dcpPacket, const uint8_t * ciphertext, uint8_t * plaintext, size_t size )`

Puts AES ECB decrypt dcpPacket to DCP input job ring.

## DCP AES non-blocking driver

### Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request.
out	<i>dcpPacket</i>	Memory for the DCP work packet.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.

### Returns

kStatus\_Success The work packet has been scheduled at DCP channel.

kStatus\_DCP\_Again The DCP channel is busy processing previous request.

**14.11.2.3** `status_t DCP_AES_EncryptCbcNonBlocking ( DCP_Type * base, dcp_handle_t * handle, dcp_work_packet_t * dcpPacket, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * iv )`

Puts AES CBC encrypt dcpPacket to DCP input job ring.

### Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>dcpPacket</i>	Memory for the DCP work packet.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.

### Returns

kStatus\_Success The work packet has been scheduled at DCP channel.

kStatus\_DCP\_Again The DCP channel is busy processing previous request.

**14.11.2.4** `status_t DCP_AES_DecryptCbcNonBlocking ( DCP_Type * base, dcp_handle_t * handle, dcp_work_packet_t * dcpPacket, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv )`

Puts AES CBC decrypt dcpPacket to DCP input job ring.

## Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>dcpPacket</i>	Memory for the DCP work packet.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.

## Returns

kStatus\_Success The work packet has been scheduled at DCP channel.

kStatus\_DCP\_Again The DCP channel is busy processing previous request.

## DCP HASH driver

### 14.12 DCP HASH driver

#### 14.12.1 Overview

This section describes the programming interface of the DCP HASH driver.

#### Data Structures

- struct [dcp\\_hash\\_ctx\\_t](#)  
*Storage type used to save hash context. [More...](#)*

#### Macros

- #define [DCP\\_SHA\\_BLOCK\\_SIZE](#) 128  
*DCP HASH Context size.*
- #define [DCP\\_HASH\\_BLOCK\\_SIZE](#) [DCP\\_SHA\\_BLOCK\\_SIZE](#)  
*DCP hash block size.*
- #define [DCP\\_HASH\\_CTX\\_SIZE](#) 58  
*DCP HASH Context size.*

#### Enumerations

- enum [dcp\\_hash\\_algo\\_t](#) {  
[kDCP\\_Sha1](#),  
[kDCP\\_Sha256](#),  
[kDCP\\_Crc32](#) }  
*Supported cryptographic block cipher functions for HASH creation.*

#### Functions

- status\_t [DCP\\_HASH\\_Init](#) (DCP\_Type \*base, [dcp\\_handle\\_t](#) \*handle, [dcp\\_hash\\_ctx\\_t](#) \*ctx, [dcp\\_hash\\_algo\\_t](#) algo)  
*Initialize HASH context.*
- status\_t [DCP\\_HASH\\_Update](#) (DCP\_Type \*base, [dcp\\_hash\\_ctx\\_t](#) \*ctx, const uint8\_t \*input, size\_t inputSize)  
*Add data to current HASH.*
- status\_t [DCP\\_HASH\\_Finish](#) (DCP\_Type \*base, [dcp\\_hash\\_ctx\\_t](#) \*ctx, uint8\_t \*output, size\_t \*outputSize)  
*Finalize hashing.*
- status\_t [DCP\\_HASH](#) (DCP\_Type \*base, [dcp\\_handle\\_t](#) \*handle, [dcp\\_hash\\_algo\\_t](#) algo, const uint8\_t \*input, size\_t inputSize, uint8\_t \*output, size\_t \*outputSize)  
*Create HASH on given data.*

## 14.12.2 Data Structure Documentation

### 14.12.2.1 struct dcp\_hash\_ctx\_t

## 14.12.3 Macro Definition Documentation

### 14.12.3.1 #define DCP\_SHA\_BLOCK\_SIZE 128

internal buffer block size

### 14.12.3.2 #define DCP\_HASH\_CTX\_SIZE 58

## 14.12.4 Enumeration Type Documentation

### 14.12.4.1 enum dcp\_hash\_algo\_t

Enumerator

```
kDCP_Sha1  SHA_1.
kDCP_Sha256  SHA_256.
kDCP_Crc32  CRC_32.
```

## 14.12.5 Function Documentation

### 14.12.5.1 status\_t DCP\_HASH\_Init ( DCP\_Type \* *base*, dcp\_handle\_t \* *handle*, dcp\_hash\_ctx\_t \* *ctx*, dcp\_hash\_algo\_t *algo* )

This function initializes the HASH.

Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Specifies the DCP channel used for hashing.
out	<i>ctx</i>	Output hash context
	<i>algo</i>	Underlying algorithm to use for hash computation.

Returns

Status of initialization

## DCP HASH driver

### 14.12.5.2 **status\_t DCP\_HASH\_Update ( DCP\_Type \* *base*, dcp\_hash\_ctx\_t \* *ctx*, const uint8\_t \* *input*, size\_t *inputSize* )**

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus\_Success, the running hash has been updated (DCP has processed the input data), so the memory at input pointer can be released back to system. The DCP context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

	<i>base</i>	DCP peripheral base address
<i>in, out</i>	<i>ctx</i>	HASH context
	<i>input</i>	Input data
	<i>inputSize</i>	Size of input data in bytes

Returns

Status of the hash update operation

### 14.12.5.3 **status\_t DCP\_HASH\_Finish ( DCP\_Type \* *base*, dcp\_hash\_ctx\_t \* *ctx*, uint8\_t \* *output*, size\_t \* *outputSize* )**

Outputs the final hash (computed by [DCP\\_HASH\\_Update\(\)](#)) and erases the context.

Parameters

<i>in, out</i>	<i>ctx</i>	Input hash context
<i>out</i>	<i>output</i>	Output hash data
<i>in, out</i>	<i>outputSize</i>	Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes.

Returns

Status of the hash finish operation

### 14.12.5.4 **status\_t DCP\_HASH ( DCP\_Type \* *base*, dcp\_handle\_t \* *handle*, dcp\_hash\_algo\_t *algo*, const uint8\_t \* *input*, size\_t *inputSize*, uint8\_t \* *output*, size\_t \* *outputSize* )**

Perform the full SHA or CRC32 in one function call. The function is blocking.



## Parameters

	<i>base</i>	DCP peripheral base address
	<i>handle</i>	Handle used for the request.
	<i>algo</i>	Underlying algorithm to use for hash computation.
	<i>input</i>	Input data
	<i>inputSize</i>	Size of input data in bytes
out	<i>output</i>	Output hash data
out	<i>outputSize</i>	Output parameter storing the size of the output hash in bytes

## Returns

Status of the one call hash operation.



# Chapter 15

## DMAMUX: Direct Memory Access Multiplexer Driver

### 15.1 Overview

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

### 15.2 Typical use case

#### 15.2.1 DMAMUX Operation

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

#### Driver version

- `#define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`  
*DMAMUX driver version 2.0.2.*

#### DMAMUX Initialization and de-initialization

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

#### DMAMUX Channel Operation

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

### 15.3 Macro Definition Documentation

#### 15.3.1 `#define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

---

## Function Documentation

### 15.4 Function Documentation

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

This function ungates the DMAMUX clock.

Parameters

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

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

This function gates the DMAMUX clock.

Parameters

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

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

This function enables the DMAMUX channel.

Parameters

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

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

This function disables the DMAMUX channel.

Note

The user must disable the DMAMUX channel before configuring it.

Parameters

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

## Function Documentation

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

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

Parameters

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

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

This function enables the DMAMUX period trigger feature.

Parameters

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

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

This function disables the DMAMUX period trigger.

Parameters

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

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

This function enables the DMAMUX channel always ON feature.

## Parameters

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





# Chapter 16

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

### 16.1 Overview

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

### 16.2 Typical use case

#### 16.2.1 eDMA Operation

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

### Data Structures

- struct `edma_config_t`  
*eDMA global configuration structure. [More...](#)*
- struct `edma_transfer_config_t`  
*eDMA transfer configuration [More...](#)*
- struct `edma_channel_preemption_config_t`  
*eDMA channel priority configuration [More...](#)*
- struct `edma_minor_offset_config_t`  
*eDMA minor offset configuration [More...](#)*
- struct `edma_tcd_t`  
*eDMA TCD. [More...](#)*
- struct `edma_handle_t`  
*eDMA transfer handle structure [More...](#)*

### Macros

- #define `DMA_DCHPRI_INDEX(channel)`  $((channel) \& \sim 0x03U) | (3 - ((channel) \& 0x03U))$   
*Compute the offset unit from DCHPRI3.*
- #define `DMA_DCHPRIin(base, channel)`  $((volatile \text{uint8}_t *) \& ((base) \rightarrow DCHPRI3)) [DMA\_DCHPRI\_INDEX(channel)]$   
*Get the pointer of DCHPRIin.*

### Typedefs

- typedef void(\* `edma_callback` )(struct `_edma_handle` \*handle, void \*userData, bool transferDone, `uint32_t` tcds)  
*Define callback function for eDMA.*

## Typical use case

### Enumerations

- enum `edma_transfer_size_t` {  
    `kEDMA_TransferSize1Bytes` = 0x0U,  
    `kEDMA_TransferSize2Bytes` = 0x1U,  
    `kEDMA_TransferSize4Bytes` = 0x2U,  
    `kEDMA_TransferSize8Bytes` = 0x3U,  
    `kEDMA_TransferSize16Bytes` = 0x4U,  
    `kEDMA_TransferSize32Bytes` = 0x5U }  
    *eDMA transfer configuration*
- enum `edma_modulo_t` {  
    `kEDMA_ModuloDisable` = 0x0U,  
    `kEDMA_Modulo2bytes`,  
    `kEDMA_Modulo4bytes`,  
    `kEDMA_Modulo8bytes`,  
    `kEDMA_Modulo16bytes`,  
    `kEDMA_Modulo32bytes`,  
    `kEDMA_Modulo64bytes`,  
    `kEDMA_Modulo128bytes`,  
    `kEDMA_Modulo256bytes`,  
    `kEDMA_Modulo512bytes`,  
    `kEDMA_Modulo1Kbytes`,  
    `kEDMA_Modulo2Kbytes`,  
    `kEDMA_Modulo4Kbytes`,  
    `kEDMA_Modulo8Kbytes`,  
    `kEDMA_Modulo16Kbytes`,  
    `kEDMA_Modulo32Kbytes`,  
    `kEDMA_Modulo64Kbytes`,  
    `kEDMA_Modulo128Kbytes`,  
    `kEDMA_Modulo256Kbytes`,  
    `kEDMA_Modulo512Kbytes`,  
    `kEDMA_Modulo1Mbytes`,  
    `kEDMA_Modulo2Mbytes`,  
    `kEDMA_Modulo4Mbytes`,  
    `kEDMA_Modulo8Mbytes`,  
    `kEDMA_Modulo16Mbytes`,  
    `kEDMA_Modulo32Mbytes`,  
    `kEDMA_Modulo64Mbytes`,  
    `kEDMA_Modulo128Mbytes`,  
    `kEDMA_Modulo256Mbytes`,  
    `kEDMA_Modulo512Mbytes`,  
    `kEDMA_Modulo1Gbytes`,  
    `kEDMA_Modulo2Gbytes` }  
    *eDMA modulo configuration*
- enum `edma_bandwidth_t` {

```
kEDMA_BandwidthStallNone = 0x0U,
kEDMA_BandwidthStall4Cycle = 0x2U,
kEDMA_BandwidthStall8Cycle = 0x3U }
```

*Bandwidth control.*

- enum `edma_channel_link_type_t` {
 

```
kEDMA_LinkNone = 0x0U,
kEDMA_MinorLink,
kEDMA_MajorLink }
```

*Channel link type.*

- enum `_edma_channel_status_flags` {
 

```
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, 1, 2))  
*eDMA driver version*

## Typical use case

### eDMA initialization and de-initialization

- void [EDMA\\_Init](#) (DMA\_Type \*base, const [edma\\_config\\_t](#) \*config)  
*Initializes the eDMA peripheral.*
- void [EDMA\\_Deinit](#) (DMA\_Type \*base)  
*Deinitializes the eDMA peripheral.*
- void [EDMA\\_InstallTCD](#) (DMA\_Type \*base, uint32\_t channel, [edma\\_tcd\\_t](#) \*tcd)  
*Push content of TCD structure into hardware TCD register.*
- void [EDMA\\_GetDefaultConfig](#) ([edma\\_config\\_t](#) \*config)  
*Gets the eDMA default configuration structure.*

### eDMA Channel Operation

- void [EDMA\\_ResetChannel](#) (DMA\_Type \*base, uint32\_t channel)  
*Sets all TCD registers to default values.*
- void [EDMA\\_SetTransferConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_transfer\\_config\\_t](#) \*config, [edma\\_tcd\\_t](#) \*nextTcd)  
*Configures the eDMA transfer attribute.*
- void [EDMA\\_SetMinorOffsetConfig](#) (DMA\_Type \*base, uint32\_t channel, const [edma\\_minor\\_offset\\_config\\_t](#) \*config)  
*Configures the eDMA minor offset feature.*
- 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 the destination modulo for the eDMA transfer.*
- static void [EDMA\\_EnableAsyncRequest](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an async request for the eDMA transfer.*
- static void [EDMA\\_EnableAutoStopRequest](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an auto stop request for the eDMA transfer.*
- void [EDMA\\_EnableChannelInterrupts](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Enables the interrupt source for the eDMA transfer.*
- void [EDMA\\_DisableChannelInterrupts](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Disables the interrupt source for the eDMA transfer.*

### eDMA TCD Operation

- void [EDMA\\_TcdReset](#) ([edma\\_tcd\\_t](#) \*tcd)  
*Sets all fields to default values for the TCD structure.*
- void [EDMA\\_TcdSetTransferConfig](#) ([edma\\_tcd\\_t](#) \*tcd, const [edma\\_transfer\\_config\\_t](#) \*config, [edma\\_tcd\\_t](#) \*nextTcd)  
*Configures the eDMA TCD transfer attribute.*
- void [EDMA\\_TcdSetMinorOffsetConfig](#) ([edma\\_tcd\\_t](#) \*tcd, const [edma\\_minor\\_offset\\_config\\_t](#) \*config)

- *Configures the eDMA TCD minor offset feature.*
- void [EDMA\\_TcdSetChannelLink](#) (edma\_tcd\_t \*tcd, edma\_channel\_link\_type\_t type, uint32\_t linkedChannel)
  - *Sets the channel link for the eDMA TCD.*
- static void [EDMA\\_TcdSetBandWidth](#) (edma\_tcd\_t \*tcd, edma\_bandwidth\_t bandWidth)
  - *Sets the bandwidth for the eDMA TCD.*
- void [EDMA\\_TcdSetModulo](#) (edma\_tcd\_t \*tcd, edma\_modulo\_t srcModulo, edma\_modulo\_t destModulo)
  - *Sets the source modulo and the destination modulo for the eDMA TCD.*
- static void [EDMA\\_TcdEnableAutoStopRequest](#) (edma\_tcd\_t \*tcd, bool enable)
  - *Sets the auto stop request for the eDMA TCD.*
- void [EDMA\\_TcdEnableInterrupts](#) (edma\_tcd\_t \*tcd, uint32\_t mask)
  - *Enables the interrupt source for the eDMA TCD.*
- void [EDMA\\_TcdDisableInterrupts](#) (edma\_tcd\_t \*tcd, uint32\_t mask)
  - *Disables the interrupt source for the eDMA TCD.*

## eDMA Channel Transfer Operation

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

## eDMA Channel Status Operation

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

## eDMA Transactional Operation

- void [EDMA\\_CreateHandle](#) (edma\_handle\_t \*handle, DMA\_Type \*base, uint32\_t channel)
  - *Creates the eDMA handle.*
- void [EDMA\\_InstallTCDMemory](#) (edma\_handle\_t \*handle, edma\_tcd\_t \*tcdPool, uint32\_t tcdSize)
  - *Installs the TCDs memory pool into the eDMA handle.*
- void [EDMA\\_SetCallback](#) (edma\_handle\_t \*handle, edma\_callback callback, void \*userData)
  - *Installs a callback function for the eDMA transfer.*
- void [EDMA\\_PrepareTransfer](#) (edma\_transfer\_config\_t \*config, void \*srcAddr, uint32\_t srcWidth, void \*destAddr, uint32\_t destWidth, uint32\_t bytesEachRequest, uint32\_t transferBytes, edma\_transfer\_type\_t type)
  - *Prepares the eDMA transfer structure.*
- status\_t [EDMA\\_SubmitTransfer](#) (edma\_handle\_t \*handle, const edma\_transfer\_config\_t \*config)
  - *Submits the eDMA transfer request.*

## Data Structure Documentation

- void [EDMA\\_StartTransfer](#) ([edma\\_handle\\_t](#) \*handle)  
*eDMA starts transfer.*
- void [EDMA\\_StopTransfer](#) ([edma\\_handle\\_t](#) \*handle)  
*eDMA stops transfer.*
- void [EDMA\\_AbortTransfer](#) ([edma\\_handle\\_t](#) \*handle)  
*eDMA aborts transfer.*
- static uint32\_t [EDMA\\_GetUnusedTCDNumber](#) ([edma\\_handle\\_t](#) \*handle)  
*Get unused TCD slot number.*
- static uint32\_t [EDMA\\_GetNextTCDAddress](#) ([edma\\_handle\\_t](#) \*handle)  
*Get the next tcd address.*
- void [EDMA\\_HandleIRQ](#) ([edma\\_handle\\_t](#) \*handle)  
*eDMA IRQ handler for the current major loop transfer completion.*

## 16.3 Data Structure Documentation

### 16.3.1 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.*

#### 16.3.1.0.8 Field Documentation

##### 16.3.1.0.8.1 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.

##### 16.3.1.0.8.2 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.

##### 16.3.1.0.8.3 bool [edma\\_config\\_t::enableDebugMode](#)

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

### 16.3.2 struct edma\_transfer\_config\_t

This structure configures the source/destination transfer attribute.

#### Data Fields

- uint32\_t [srcAddr](#)  
*Source data address.*
- uint32\_t [destAddr](#)  
*Destination data address.*
- [edma\\_transfer\\_size\\_t srcTransferSize](#)  
*Source data transfer size.*
- [edma\\_transfer\\_size\\_t destTransferSize](#)  
*Destination data transfer size.*
- int16\_t [srcOffset](#)  
*Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.*
- int16\_t [destOffset](#)  
*Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.*
- uint32\_t [minorLoopBytes](#)  
*Bytes to transfer in a minor loop.*
- uint32\_t [majorLoopCounts](#)  
*Major loop iteration count.*

#### 16.3.2.0.0.9 Field Documentation

16.3.2.0.0.9.1 uint32\_t edma\_transfer\_config\_t::srcAddr

16.3.2.0.0.9.2 uint32\_t edma\_transfer\_config\_t::destAddr

16.3.2.0.0.9.3 edma\_transfer\_size\_t edma\_transfer\_config\_t::srcTransferSize

16.3.2.0.0.9.4 edma\_transfer\_size\_t edma\_transfer\_config\_t::destTransferSize

16.3.2.0.0.9.5 int16\_t edma\_transfer\_config\_t::srcOffset

16.3.2.0.0.9.6 int16\_t edma\_transfer\_config\_t::destOffset

16.3.2.0.0.9.7 uint32\_t edma\_transfer\_config\_t::majorLoopCounts

### 16.3.3 struct edma\_channel\_Preemption\_config\_t

#### Data Fields

- bool [enableChannelPreemption](#)  
*If true: a channel can be suspended by other channel with higher priority.*
- bool [enablePreemptAbility](#)

## Data Structure Documentation

- *If true: a channel can suspend other channel with low priority.*  
• uint8\_t **channelPriority**  
*Channel priority.*

### 16.3.4 struct edma\_minor\_offset\_config\_t

#### Data Fields

- bool **enableSrcMinorOffset**  
*Enable(true) or Disable(false) source minor loop offset.*
- bool **enableDestMinorOffset**  
*Enable(true) or Disable(false) destination minor loop offset.*
- uint32\_t **minorOffset**  
*Offset for a minor loop mapping.*

#### 16.3.4.0.0.10 Field Documentation

16.3.4.0.0.10.1 bool edma\_minor\_offset\_config\_t::enableSrcMinorOffset

16.3.4.0.0.10.2 bool edma\_minor\_offset\_config\_t::enableDestMinorOffset

16.3.4.0.0.10.3 uint32\_t edma\_minor\_offset\_config\_t::minorOffset

### 16.3.5 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.*

#### 16.3.5.0.0.11 Field Documentation

16.3.5.0.0.11.1 `__IO uint16_t edma_tcd_t::CITER`

16.3.5.0.0.11.2 `__IO uint16_t edma_tcd_t::BITER`

### 16.3.6 struct edma\_handle\_t

#### Data Fields

- `edma_callback callback`  
*Callback function for major count exhausted.*
- `void * userData`  
*Callback function parameter.*
- `DMA_Type * base`  
*eDMA peripheral base address.*
- `edma_tcd_t * tcdPool`  
*Pointer to memory stored TCDs.*
- `uint8_t channel`  
*eDMA channel number.*
- `volatile int8_t header`  
*The first TCD index.*
- `volatile int8_t tail`  
*The last TCD index.*
- `volatile int8_t tcdUsed`  
*The number of used TCD slots.*
- `volatile int8_t tcdSize`  
*The total number of TCD slots in the queue.*
- `uint8_t flags`  
*The status of the current channel.*

## Typedef Documentation

### 16.3.6.0.0.12 Field Documentation

**16.3.6.0.0.12.1** `edma_callback edma_handle_t::callback`

**16.3.6.0.0.12.2** `void* edma_handle_t::userData`

**16.3.6.0.0.12.3** `DMA_Type* edma_handle_t::base`

**16.3.6.0.0.12.4** `edma_tcd_t* edma_handle_t::tcdPool`

**16.3.6.0.0.12.5** `uint8_t edma_handle_t::channel`

**16.3.6.0.0.12.6** `volatile int8_t edma_handle_t::header`

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

**16.3.6.0.0.12.7** `volatile int8_t edma_handle_t::tail`

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

**16.3.6.0.0.12.8** `volatile int8_t edma_handle_t::tcdUsed`

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

**16.3.6.0.0.12.9** `volatile int8_t edma_handle_t::tcdSize`

**16.3.6.0.0.12.10** `uint8_t edma_handle_t::flags`

## 16.4 Macro Definition Documentation

**16.4.1** `#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))`

Version 2.1.2.

## 16.5 Typedef Documentation

**16.5.1** `typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcDs)`

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

### Parameters

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

## 16.6 Enumeration Type Documentation

### 16.6.1 enum edma\_transfer\_size\_t

#### Enumerator

- kEDMA\_TransferSize1Bytes* Source/Destination data transfer size is 1 byte every time.
- kEDMA\_TransferSize2Bytes* Source/Destination data transfer size is 2 bytes every time.
- kEDMA\_TransferSize4Bytes* Source/Destination data transfer size is 4 bytes every time.
- kEDMA\_TransferSize8Bytes* Source/Destination data transfer size is 8 bytes every time.
- kEDMA\_TransferSize16Bytes* Source/Destination data transfer size is 16 bytes every time.
- kEDMA\_TransferSize32Bytes* Source/Destination data transfer size is 32 bytes every time.

### 16.6.2 enum edma\_modulo\_t

#### Enumerator

- kEDMA\_ModuloDisable* Disable modulo.
- kEDMA\_Modulo2bytes* Circular buffer size is 2 bytes.
- kEDMA\_Modulo4bytes* Circular buffer size is 4 bytes.
- kEDMA\_Modulo8bytes* Circular buffer size is 8 bytes.
- kEDMA\_Modulo16bytes* Circular buffer size is 16 bytes.
- kEDMA\_Modulo32bytes* Circular buffer size is 32 bytes.
- kEDMA\_Modulo64bytes* Circular buffer size is 64 bytes.
- kEDMA\_Modulo128bytes* Circular buffer size is 128 bytes.
- kEDMA\_Modulo256bytes* Circular buffer size is 256 bytes.
- kEDMA\_Modulo512bytes* Circular buffer size is 512 bytes.
- kEDMA\_Modulo1Kbytes* Circular buffer size is 1 K bytes.

## Enumeration Type Documentation

*kEDMA\_Modulo2Kbytes* Circular buffer size is 2 K bytes.  
*kEDMA\_Modulo4Kbytes* Circular buffer size is 4 K bytes.  
*kEDMA\_Modulo8Kbytes* Circular buffer size is 8 K bytes.  
*kEDMA\_Modulo16Kbytes* Circular buffer size is 16 K bytes.  
*kEDMA\_Modulo32Kbytes* Circular buffer size is 32 K bytes.  
*kEDMA\_Modulo64Kbytes* Circular buffer size is 64 K bytes.  
*kEDMA\_Modulo128Kbytes* Circular buffer size is 128 K bytes.  
*kEDMA\_Modulo256Kbytes* Circular buffer size is 256 K bytes.  
*kEDMA\_Modulo512Kbytes* Circular buffer size is 512 K bytes.  
*kEDMA\_Modulo1Mbytes* Circular buffer size is 1 M bytes.  
*kEDMA\_Modulo2Mbytes* Circular buffer size is 2 M bytes.  
*kEDMA\_Modulo4Mbytes* Circular buffer size is 4 M bytes.  
*kEDMA\_Modulo8Mbytes* Circular buffer size is 8 M bytes.  
*kEDMA\_Modulo16Mbytes* Circular buffer size is 16 M bytes.  
*kEDMA\_Modulo32Mbytes* Circular buffer size is 32 M bytes.  
*kEDMA\_Modulo64Mbytes* Circular buffer size is 64 M bytes.  
*kEDMA\_Modulo128Mbytes* Circular buffer size is 128 M bytes.  
*kEDMA\_Modulo256Mbytes* Circular buffer size is 256 M bytes.  
*kEDMA\_Modulo512Mbytes* Circular buffer size is 512 M bytes.  
*kEDMA\_Modulo1Gbytes* Circular buffer size is 1 G bytes.  
*kEDMA\_Modulo2Gbytes* Circular buffer size is 2 G bytes.

### 16.6.3 enum edma\_bandwidth\_t

Enumerator

*kEDMA\_BandwidthStallNone* No eDMA engine stalls.  
*kEDMA\_BandwidthStall4Cycle* eDMA engine stalls for 4 cycles after each read/write.  
*kEDMA\_BandwidthStall8Cycle* eDMA engine stalls for 8 cycles after each read/write.

### 16.6.4 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.

### 16.6.5 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

### 16.6.6 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.

### 16.6.7 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.

### 16.6.8 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.

### 16.6.9 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.

---

## Function Documentation

### 16.7 Function Documentation

#### 16.7.1 void EDMA\_Init ( DMA\_Type \* *base*, const edma\_config\_t \* *config* )

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure.

## Parameters

<i>base</i>	eDMA peripheral base address.
<i>config</i>	A pointer to the configuration structure, see "edma_config_t".

## Note

This function enables the minor loop map feature.

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

This function gates the eDMA clock.

## Parameters

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

### 16.7.3 void EDMA\_InstallTCD ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_tcd\_t \* *tcd* )

## Parameters

<i>base</i>	EDMA peripheral base address.
<i>channel</i>	EDMA channel number.
<i>tcd</i>	Point to TCD structure.

### 16.7.4 void EDMA\_GetDefaultConfig ( edma\_config\_t \* *config* )

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
*  config.enableContinuousLinkMode = false;
*  config.enableHaltOnError = true;
*  config.enableRoundRobinArbitration = false;
*  config.enableDebugMode = false;
*
```

## Function Documentation

Parameters

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

### 16.7.5 void EDMA\_ResetChannel ( DMA\_Type \* *base*, uint32\_t *channel* )

This function sets TCD registers for this channel to default values.

Parameters

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

Note

This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

This function enables the auto stop request feature.

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

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

Example:

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

Parameters

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



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

## Note

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

### 16.7.7 void EDMA\_SetMinorOffsetConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_minor\_offset\_config\_t \* *config* )

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

## Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>config</i>	A pointer to the minor offset configuration structure.

### 16.7.8 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>	A pointer to the channel preemption configuration structure.

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

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

---

## Function Documentation

triggered when the CITER is exhausted.

## Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>type</i>	A channel link type, which can be one of the following: <ul style="list-style-type: none"> <li>• kEDMA_LinkNone</li> <li>• kEDMA_MinorLink</li> <li>• kEDMA_MajorLink</li> </ul>
<i>linkedChannel</i>	The linked channel number.

## Note

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

### 16.7.10 void EDMA\_SetBandWidth ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_bandwidth\_t *bandWidth* )

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

## Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>bandWidth</i>	A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"> <li>• kEDMABandwidthStallNone</li> <li>• kEDMABandwidthStall4Cycle</li> <li>• kEDMABandwidthStall8Cycle</li> </ul>

### 16.7.11 void EDMA\_SetModulo ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_modulo\_t *srcModulo*, edma\_modulo\_t *destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

## Function Documentation

### Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>srcModulo</i>	A source modulo value.
<i>destModulo</i>	A destination modulo value.

**16.7.12 static void EDMA\_EnableAsyncRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]**

### Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>enable</i>	The command to enable (true) or disable (false).

**16.7.13 static void EDMA\_EnableAutoStopRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]**

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

### Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>enable</i>	The command to enable (true) or disable (false).

**16.7.14 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.

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

## Function Documentation

### Parameters

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

### 16.7.16 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.

### 16.7.17 void EDMA\_TcdSetTransferConfig ( edma\_tcd\_t \* *tcd*, const edma\_transfer\_config\_t \* *config*, edma\_tcd\_t \* *nextTcd* )

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The STCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

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

### Parameters

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

## Note

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

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

### 16.7.18 void EDMA\_TcdSetMinorOffsetConfig ( edma\_tcd\_t \* *tcd*, const edma\_minor\_offset\_config\_t \* *config* )

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

## Parameters

<i>tcd</i>	A point to the TCD structure.
<i>config</i>	A pointer to the minor offset configuration structure.

### 16.7.19 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.

## Function Documentation

### 16.7.20 `static void EDMA_TcdSetBandWidth ( edma_tcd_t * tcd, edma_bandwidth_t bandWidth ) [inline], [static]`

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

<i>tcd</i>	A pointer to the TCD structure.
<i>bandWidth</i>	A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"><li>• kEDMABandwidthStallNone</li><li>• kEDMABandwidthStall4Cycle</li><li>• kEDMABandwidthStall8Cycle</li></ul>

### 16.7.21 `void EDMA_TcdSetModulo ( edma_tcd_t * tcd, edma_modulo_t srcModulo, edma_modulo_t destModulo )`

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

<i>tcd</i>	A pointer to the TCD structure.
<i>srcModulo</i>	A source modulo value.
<i>destModulo</i>	A destination modulo value.

### 16.7.22 `static void EDMA_TcdEnableAutoStopRequest ( edma_tcd_t * tcd, bool enable ) [inline], [static]`

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

<i>tcd</i>	A pointer to the TCD structure.
<i>enable</i>	The command to enable (true) or disable (false).



16.7.23 void EDMA\_TcdEnableInterrupts ( edma\_tcd\_t \* *tcd*, uint32\_t *mask* )

## Function Documentation

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>mask</i>	The mask of interrupt source to be set. Users need to use the defined <code>edma_interrupt_enable_t</code> type.

### 16.7.24 void EDMA\_TcdDisableInterrupts ( edma\_tcd\_t \* *tcd*, uint32\_t *mask* )

Parameters

<i>tcd</i>	Point to the TCD structure.
<i>mask</i>	The mask of interrupt source to be set. Users need to use the defined <code>edma_interrupt_enable_t</code> type.

### 16.7.25 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.

### 16.7.26 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.

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

This function starts a minor loop transfer.

## Function Documentation

### Parameters

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

### 16.7.28 `uint32_t EDMA_GetRemainingMajorLoopCount ( DMA_Type * base, uint32_t channel )`

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

### Parameters

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

### Returns

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

### Note

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

### 16.7.29 `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.

### 16.7.30 `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.

### 16.7.31 `void EDMA_ClearChannelStatusFlags ( DMA_Type * base, uint32_t channel, uint32_t mask )`

## Parameters

<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.
<i>mask</i>	The mask of channel status to be cleared. Users need to use the defined <code>_edma_channel_status_flags</code> type.

### 16.7.32 `void EDMA_CreateHandle ( edma_handle_t * handle, DMA_Type * base, uint32_t channel )`

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

## Function Documentation

### Parameters

<i>handle</i>	eDMA handle pointer. The eDMA handle stores callback function and parameters.
<i>base</i>	eDMA peripheral base address.
<i>channel</i>	eDMA channel number.

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

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

### Parameters

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

### 16.7.34 void EDMA\_SetCallback ( edma\_handle\_t \* *handle*, edma\_callback *callback*, void \* *userData* )

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

### Parameters

<i>handle</i>	eDMA handle pointer.
<i>callback</i>	eDMA callback function pointer.
<i>userData</i>	A parameter for the callback function.

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

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

## Parameters

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

## Note

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

### 16.7.36 `status_t EDMA_SubmitTransfer ( edma_handle_t * handle, const edma_transfer_config_t * config )`

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function `EDMA_InstallTCDMemory` before.

## Parameters

<i>handle</i>	eDMA handle pointer.
<i>config</i>	Pointer to eDMA transfer configuration structure.

## Return values

<i>kStatus_EDMA_Success</i>	It means submit transfer request succeed.
<i>kStatus_EDMA_Queue-Full</i>	It means TCD queue is full. Submit transfer request is not allowed.

## Function Documentation

<i>kStatus_EDMA_Busy</i>	It means the given channel is busy, need to submit request later.
--------------------------	-------------------------------------------------------------------

### 16.7.37 void EDMA\_StartTransfer ( edma\_handle\_t \* handle )

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

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

### 16.7.38 void EDMA\_StopTransfer ( edma\_handle\_t \* handle )

This function disables the channel request to pause the transfer. Users can call [EDMA\\_StartTransfer\(\)](#) again to resume the transfer.

Parameters

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

### 16.7.39 void EDMA\_AbortTransfer ( edma\_handle\_t \* handle )

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

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

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

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.



## Parameters

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

## Returns

The unused tcd slot number.

### 16.7.41 `static uint32_t EDMA_GetNextTCDAddress ( edma_handle_t * handle )` `[inline], [static]`

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

## Parameters

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

## Returns

The next TCD address.

### 16.7.42 `void EDMA_HandleIRQ ( edma_handle_t * handle )`

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga\_index are calculated based on the DLAST\_SGA bitfield lies in the TCD\_CSR register, the sga\_index in this case should be 2 (DLAST\_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description part of the Reference Manual for further details.

---

## Function Documentation

Parameters

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

## Chapter 17

# eLCDIF: Enhanced LCD Interface

### 17.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Enhanced LCD Interface(eLCDIF)

The Enhanced LCD Interface supports MPU mode, VSYNC mode, RGB mode (or DOTCLK mode), and DVI mode. The current eLCDIF driver only supports RGB mode.

### 17.2 Typical use case

#### 17.2.1 Frame buffer update

The function `ELCDIF_SetNextBufferAddr` sets the next frame to show to eLCDIF, the eLCDIF loads the new frame and sets the interrupt `kELCDIF_CurFrameDone`. If no new frame is set, the old one is displayed.

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

#### 17.2.2 Alpha surface

The alpha surface can be enabled to add an extra overlay on the normal display buffer. In this example, the alpha surface is enabled, and the alpha value is updated after every frame loaded to eLCDIF.

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

### Data Structures

- struct `elcdif_pixel_format_reg_t`  
*The register value when using different pixel format. [More...](#)*
- struct `elcdif_rgb_mode_config_t`  
*eLCDIF configure structure for RGB mode (DOTCLK mode). [More...](#)*
- struct `elcdif_as_buffer_config_t`  
*eLCDIF alpha surface buffer configuration. [More...](#)*
- struct `elcdif_as_blend_config_t`  
*eLCDIF alpha surface blending configuration. [More...](#)*

## Typical use case

### Enumerations

- enum `_elcdif_polarity_flags` {  
    `kELCDIF_VsyncActiveLow` = 0U,  
    `kELCDIF_VsyncActiveHigh` = LCDIF\_VDCTRL0\_VSYNC\_POL\_MASK,  
    `kELCDIF_HsyncActiveLow` = 0U,  
    `kELCDIF_HsyncActiveHigh` = LCDIF\_VDCTRL0\_HSYNC\_POL\_MASK,  
    `kELCDIF_DataEnableActiveLow` = 0U,  
    `kELCDIF_DataEnableActiveHigh` = LCDIF\_VDCTRL0\_ENABLE\_POL\_MASK,  
    `kELCDIF_DriveDataOnFallingClkEdge` = 0U,  
    `kELCDIF_DriveDataOnRisingClkEdge` = LCDIF\_VDCTRL0\_DOTCLK\_POL\_MASK }  
    *eLCDIF signal polarity flags*
- enum `_elcdif_interrupt_enable` {  
    `kELCDIF_BusMasterErrorInterruptEnable` = LCDIF\_CTRL1\_BM\_ERROR\_IRQ\_EN\_MASK,  
    `kELCDIF_TxFifoOverflowInterruptEnable` = LCDIF\_CTRL1\_OVERFLOW\_IRQ\_EN\_MASK,  
    `kELCDIF_TxFifoUnderflowInterruptEnable` = LCDIF\_CTRL1\_UNDERFLOW\_IRQ\_EN\_MASK,  
    `kELCDIF_CurFrameDoneInterruptEnable`,  
    `kELCDIF_VsyncEdgeInterruptEnable` }  
    *The eLCDIF interrupts to enable.*
- enum `_elcdif_interrupt_flags` {  
    `kELCDIF_BusMasterError` = LCDIF\_CTRL1\_BM\_ERROR\_IRQ\_MASK,  
    `kELCDIF_TxFifoOverflow` = LCDIF\_CTRL1\_OVERFLOW\_IRQ\_MASK,  
    `kELCDIF_TxFifoUnderflow` = LCDIF\_CTRL1\_UNDERFLOW\_IRQ\_MASK,  
    `kELCDIF_CurFrameDone`,  
    `kELCDIF_VsyncEdge` = LCDIF\_CTRL1\_VSYNC\_EDGE\_IRQ\_MASK }  
    *The eLCDIF interrupt status flags.*
- enum `_elcdif_status_flags` {  
    `kELCDIF_LFifoFull` = LCDIF\_STAT\_LFIFO\_FULL\_MASK,  
    `kELCDIF_LFifoEmpty` = LCDIF\_STAT\_LFIFO\_EMPTY\_MASK,  
    `kELCDIF_TxFifoFull` = LCDIF\_STAT\_TXFIFO\_FULL\_MASK,  
    `kELCDIF_TxFifoEmpty` = LCDIF\_STAT\_TXFIFO\_EMPTY\_MASK }  
    *eLCDIF status flags*
- enum `elcdif_pixel_format_t` {  
    `kELCDIF_PixelFormatRAW8` = 0,  
    `kELCDIF_PixelFormatRGB565` = 1,  
    `kELCDIF_PixelFormatRGB666` = 2,  
    `kELCDIF_PixelFormatXRGB8888` = 3,  
    `kELCDIF_PixelFormatRGB888` = 4 }  
    *The pixel format.*
- enum `elcdif_lcd_data_bus_t` {  
    `kELCDIF_DataBus8Bit` = LCDIF\_CTRL\_LCD\_DATABUS\_WIDTH(1),  
    `kELCDIF_DataBus16Bit` = LCDIF\_CTRL\_LCD\_DATABUS\_WIDTH(0),  
    `kELCDIF_DataBus18Bit` = LCDIF\_CTRL\_LCD\_DATABUS\_WIDTH(2),  
    `kELCDIF_DataBus24Bit` = LCDIF\_CTRL\_LCD\_DATABUS\_WIDTH(3) }  
    *The LCD data bus type.*
- enum `elcdif_as_pixel_format_t` {

```

kELCDIF_AsPixelFormatARGB8888 = 0x0,
kELCDIF_AsPixelFormatRGB888 = 0x4,
kELCDIF_AsPixelFormatARGB1555 = 0x8,
kELCDIF_AsPixelFormatARGB4444 = 0x9,
kELCDIF_AsPixelFormatRGB555 = 0xC,
kELCDIF_AsPixelFormatRGB444 = 0xD,
kELCDIF_AsPixelFormatRGB565 = 0xE }
    eLCDIF alpha surface pixel format.
• enum elcdif_alpha_mode_t {
    kELCDIF_AlphaEmbedded,
    kELCDIF_AlphaOverride,
    kELCDIF_AlphaMultiply,
    kELCDIF_AlphaRop }
    eLCDIF alpha mode during blending.
• enum elcdif_rop_mode_t {
    kELCDIF_RopMaskAs = 0x0,
    kELCDIF_RopMaskNotAs = 0x1,
    kELCDIF_RopMaskAsNot = 0x2,
    kELCDIF_RopMergeAs = 0x3,
    kELCDIF_RopMergeNotAs = 0x4,
    kELCDIF_RopMergeAsNot = 0x5,
    kELCDIF_RopNotCopyAs = 0x6,
    kELCDIF_RopNot = 0x7,
    kELCDIF_RopNotMaskAs = 0x8,
    kELCDIF_RopNotMergeAs = 0x9,
    kELCDIF_RopXorAs = 0xA,
    kELCDIF_RopNotXorAs = 0xB }
    eLCDIF ROP mode during blending.
• enum elcdif_lut_t {
    kELCDIF_Lut0 = 0,
    kELCDIF_Lut1 }
    eLCDIF LUT

```

## Driver version

- #define `FSL_ELCDIF_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*eLCDIF driver version*

## eLCDIF initialization and de-initialization

- void `ELCDIF_RgbModeInit` (LCDIF\_Type \*base, const `elcdif_rgb_mode_config_t` \*config)  
*Initializes the eLCDIF to work in RGB mode (DOTCLK mode).*
- static uint32\_t `ELCDIF_GetStatus` (LCDIF\_Type \*base)  
*Gets the eLCDIF default configuration structure for RGB (DOTCLK) mode.*
- static uint32\_t `ELCDIF_GetLFifoCount` (LCDIF\_Type \*base)  
*Get current count in Latency buffer (LFIFO).*

## Data Structure Documentation

### Interrupts

- static void [ELCDIF\\_EnableInterrupts](#) (LCDIF\_Type \*base, uint32\_t mask)  
*Enables eLCDIF interrupt requests.*
- static void [ELCDIF\\_DisableInterrupts](#) (LCDIF\_Type \*base, uint32\_t mask)  
*Disables eLCDIF interrupt requests.*
- static uint32\_t [ELCDIF\\_GetInterruptStatus](#) (LCDIF\_Type \*base)  
*Get eLCDIF interrupt pending status.*
- static void [ELCDIF\\_ClearInterruptStatus](#) (LCDIF\_Type \*base, uint32\_t mask)  
*Clear eLCDIF interrupt pending status.*

## 17.3 Data Structure Documentation

### 17.3.1 struct elcdif\_pixel\_format\_reg\_t

These register bits control the pixel format:

- CTRL[DATA\_FORMAT\_24\_BIT]
- CTRL[DATA\_FORMAT\_18\_BIT]
- CTRL[DATA\_FORMAT\_16\_BIT]
- CTRL[WORD\_LENGTH]
- CTRL1[BYTE\_PACKING\_FORMAT]

### Data Fields

- uint32\_t [regCtrl](#)  
*Value of register CTRL.*
- uint32\_t [regCtrl1](#)  
*Value of register CTRL1.*

#### 17.3.1.0.0.13 Field Documentation

17.3.1.0.0.13.1 uint32\_t elcdif\_pixel\_format\_reg\_t::regCtrl

17.3.1.0.0.13.2 uint32\_t elcdif\_pixel\_format\_reg\_t::regCtrl1

### 17.3.2 struct elcdif\_rgb\_mode\_config\_t

### Data Fields

- uint16\_t [panelWidth](#)  
*Display panel width, pixels per line.*
- uint16\_t [panelHeight](#)  
*Display panel height, how many lines per panel.*
- uint8\_t [hsw](#)  
*HSYNC pulse width.*
- uint8\_t [hfp](#)  
*Horizontal front porch.*

- `uint8_t hbp`  
*Horizontal back porch.*
- `uint8_t vsw`  
*VSYNC pulse width.*
- `uint8_t vfp`  
*Vertical front porch.*
- `uint8_t vbp`  
*Vertical back porch.*
- `uint32_t polarityFlags`  
*OR'ed value of `_elcdif_polarity_flags`, used to control the signal polarity.*
- `uint32_t bufferAddr`  
*Frame buffer address.*
- `elcdif_pixel_format_t pixelFormat`  
*Pixel format.*
- `elcdif_lcd_data_bus_t dataBus`  
*LCD data bus.*

#### 17.3.2.0.0.14 Field Documentation

- 17.3.2.0.0.14.1 `uint16_t elcdif_rgb_mode_config_t::panelWidth`
- 17.3.2.0.0.14.2 `uint16_t elcdif_rgb_mode_config_t::panelHeight`
- 17.3.2.0.0.14.3 `uint8_t elcdif_rgb_mode_config_t::hsw`
- 17.3.2.0.0.14.4 `uint8_t elcdif_rgb_mode_config_t::hfp`
- 17.3.2.0.0.14.5 `uint8_t elcdif_rgb_mode_config_t::hbp`
- 17.3.2.0.0.14.6 `uint8_t elcdif_rgb_mode_config_t::vsw`
- 17.3.2.0.0.14.7 `uint8_t elcdif_rgb_mode_config_t::vfp`
- 17.3.2.0.0.14.8 `uint8_t elcdif_rgb_mode_config_t::vbp`
- 17.3.2.0.0.14.9 `uint32_t elcdif_rgb_mode_config_t::polarityFlags`
- 17.3.2.0.0.14.10 `uint32_t elcdif_rgb_mode_config_t::bufferAddr`
- 17.3.2.0.0.14.11 `elcdif_pixel_format_t elcdif_rgb_mode_config_t::pixelFormat`
- 17.3.2.0.0.14.12 `elcdif_lcd_data_bus_t elcdif_rgb_mode_config_t::dataBus`

#### 17.3.3 struct `elcdif_as_buffer_config_t`

##### Data Fields

- `uint32_t bufferAddr`  
*Buffer address.*
- `elcdif_as_pixel_format_t pixelFormat`

## Enumeration Type Documentation

*Pixel format.*

### 17.3.3.0.0.15 Field Documentation

17.3.3.0.0.15.1 `uint32_t elcdif_as_buffer_config_t::bufferAddr`

17.3.3.0.0.15.2 `elcdif_as_pixel_format_t elcdif_as_buffer_config_t::pixelFormat`

### 17.3.4 struct `elcdif_as_blend_config_t`

#### Data Fields

- `uint8_t alpha`  
*User defined alpha value, only used when `alphaMode` is `kELCDIF_AlphaOverride` or `kELCDIF_AlphaRop`.*
- `bool invertAlpha`  
*Set true to invert the alpha.*
- `elcdif_alpha_mode_t alphaMode`  
*Alpha mode.*
- `elcdif_rop_mode_t ropMode`  
*ROP mode, only valid when `alphaMode` is `kELCDIF_AlphaRop`.*

### 17.3.4.0.0.16 Field Documentation

17.3.4.0.0.16.1 `uint8_t elcdif_as_blend_config_t::alpha`

17.3.4.0.0.16.2 `bool elcdif_as_blend_config_t::invertAlpha`

17.3.4.0.0.16.3 `elcdif_alpha_mode_t elcdif_as_blend_config_t::alphaMode`

17.3.4.0.0.16.4 `elcdif_rop_mode_t elcdif_as_blend_config_t::ropMode`

## 17.4 Macro Definition Documentation

17.4.1 `#define FSL_ELCDIF_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

Version 2.0.0.

## 17.5 Enumeration Type Documentation

### 17.5.1 enum `_elcdif_polarity_flags`

Enumerator

`kELCDIF_VsyncActiveLow` VSYNC active low.  
`kELCDIF_VsyncActiveHigh` VSYNC active high.  
`kELCDIF_HsyncActiveLow` HSYNC active low.  
`kELCDIF_HsyncActiveHigh` HSYNC active high.  
`kELCDIF_DataEnableActiveLow` Data enable line active low.



***kELCDIF\_DataEnableActiveHigh*** Data enable line active high.

***kELCDIF\_DriveDataOnFallingClkEdge*** Drive data on falling clock edge, capture data on rising clock edge.

***kELCDIF\_DriveDataOnRisingClkEdge*** Drive data on falling clock edge, capture data on rising clock edge.

### 17.5.2 enum \_elcdif\_interrupt\_enable

Enumerator

***kELCDIF\_BusMasterErrorInterruptEnable*** Bus master error interrupt.

***kELCDIF\_TxFifoOverflowInterruptEnable*** TXFIFO overflow interrupt.

***kELCDIF\_TxFifoUnderflowInterruptEnable*** TXFIFO underflow interrupt.

***kELCDIF\_CurFrameDoneInterruptEnable*** Interrupt when hardware enters vertical blanking state.

***kELCDIF\_VsyncEdgeInterruptEnable*** Interrupt when hardware encounters VSYNC edge.

### 17.5.3 enum \_elcdif\_interrupt\_flags

Enumerator

***kELCDIF\_BusMasterError*** Bus master error interrupt.

***kELCDIF\_TxFifoOverflow*** TXFIFO overflow interrupt.

***kELCDIF\_TxFifoUnderflow*** TXFIFO underflow interrupt.

***kELCDIF\_CurFrameDone*** Interrupt when hardware enters vertical blanking state.

***kELCDIF\_VsyncEdge*** Interrupt when hardware encounters VSYNC edge.

### 17.5.4 enum \_elcdif\_status\_flags

Enumerator

***kELCDIF\_LFifoFull*** LFIFO full.

***kELCDIF\_LFifoEmpty*** LFIFO empty.

***kELCDIF\_TxFifoFull*** TXFIFO full.

***kELCDIF\_TxFifoEmpty*** TXFIFO empty.

### 17.5.5 enum elcdif\_pixel\_format\_t

This enumerator should be defined together with the array `s_pixelFormatReg`. To support new pixel format, enhance this enumerator and `s_pixelFormatReg`.

## Enumeration Type Documentation

### Enumerator

***kELCDIF\_PixelFormatRAW8*** RAW 8 bit, four data use 32 bits.

***kELCDIF\_PixelFormatRGB565*** RGB565, two pixel use 32 bits.

***kELCDIF\_PixelFormatRGB666*** RGB666 unpacked, one pixel uses 32 bits, high byte unused, upper 2 bits of other bytes unused.

***kELCDIF\_PixelFormatXRGB8888*** XRGB8888 unpacked, one pixel uses 32 bits, high byte unused.

***kELCDIF\_PixelFormatRGB888*** RGB888 packed, one pixel uses 24 bits.

### 17.5.6 enum elcdif\_lcd\_data\_bus\_t

#### Enumerator

***kELCDIF\_DataBus8Bit*** 8-bit data bus.

***kELCDIF\_DataBus16Bit*** 16-bit data bus, support RGB565.

***kELCDIF\_DataBus18Bit*** 18-bit data bus, support RGB666.

***kELCDIF\_DataBus24Bit*** 24-bit data bus, support RGB888.

### 17.5.7 enum elcdif\_as\_pixel\_format\_t

#### Enumerator

***kELCDIF\_AsPixelFormatARGB8888*** 32-bit pixels with alpha.

***kELCDIF\_AsPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)

***kELCDIF\_AsPixelFormatARGB1555*** 16-bit pixels with alpha.

***kELCDIF\_AsPixelFormatARGB4444*** 16-bit pixels with alpha.

***kELCDIF\_AsPixelFormatRGB555*** 16-bit pixels without alpha.

***kELCDIF\_AsPixelFormatRGB444*** 16-bit pixels without alpha.

***kELCDIF\_AsPixelFormatRGB565*** 16-bit pixels without alpha.

### 17.5.8 enum elcdif\_alpha\_mode\_t

#### Enumerator

***kELCDIF\_AlphaEmbedded*** The alpha surface pixel alpha value will be used for blend.

***kELCDIF\_AlphaOverride*** The user defined alpha value will be used for blend directly.

***kELCDIF\_AlphaMultiply*** The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set set to 200, user defined alpha set to 100, then the result alpha is  $200 * 100 / 255$ .

***kELCDIF\_AlphaRop*** Raster operation.

### 17.5.9 enum elcdif\_rop\_mode\_t

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Enumerator

*kELCDIF\_RopMaskAs* AS AND PS.  
*kELCDIF\_RopMaskNotAs* nAS AND PS.  
*kELCDIF\_RopMaskAsNot* AS AND nPS.  
*kELCDIF\_RopMergeAs* AS OR PS.  
*kELCDIF\_RopMergeNotAs* nAS OR PS.  
*kELCDIF\_RopMergeAsNot* AS OR nPS.  
*kELCDIF\_RopNotCopyAs* nAS.  
*kELCDIF\_RopNot* nPS.  
*kELCDIF\_RopNotMaskAs* AS NAND PS.  
*kELCDIF\_RopNotMergeAs* AS NOR PS.  
*kELCDIF\_RopXorAs* AS XOR PS.  
*kELCDIF\_RopNotXorAs* AS XNOR PS.

### 17.5.10 enum elcdif\_lut\_t

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external displayer.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determines which memory to use.

Enumerator

*kELCDIF\_Lut0* LUT 0.  
*kELCDIF\_Lut1* LUT 1.

## 17.6 Function Documentation

### 17.6.1 void ELCDIF\_RgbModelnit ( LCDIF\_Type \* *base*, const elcdif\_rgb\_mode\_config\_t \* *config* )

This function ungates the eLCDIF clock and configures the eLCDIF peripheral according to the configuration structure.

## Function Documentation

### Parameters

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

### 17.6.2 static uint32\_t ELCDIF\_GetStatus ( LCDIF\_Type \* *base* ) [inline], [static]

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config->panelWidth = 480U;
config->panelHeight = 272U;
config->hsw = 41;
config->hfp = 4;
config->hbp = 8;
config->vsw = 10;
config->vfp = 4;
config->vbp = 2;
config->polarityFlags = kELCDIF_VsyncActiveLow |
                       kELCDIF_HsyncActiveLow |
                       kELCDIF_DataEnableActiveLow |
                       kELCDIF_DriveDataOnFallingClkEdge;

config->bufferAddr = 0U;
config->pixelFormat = kELCDIF_PixelFormatRGB888;
config->dataBus = kELCDIF_DataBus24Bit;
@code
*
* @param config Pointer to the eLCDIF configuration structure.
*/
void ELCDIF_RgbModeGetDefaultConfig(elcdif_rgb_mode_config_t *config);

void ELCDIF_Deinit(LCDIF_Type *base);

/* @} */

static inline void ELCDIF_RgbModeStart(LCDIF_Type *base)
{
    base->CTRL_SET = LCDIF_CTRL_RUN_MASK | LCDIF_CTRL_DOTCLK_MODE_MASK;
}

void ELCDIF_RgbModeStop(LCDIF_Type *base);

static inline void ELCDIF_SetNextBufferAddr(LCDIF_Type *base, uint32_t bufferAddr)
{
    base->NEXT_BUF = bufferAddr;
}

void ELCDIF_Reset(LCDIF_Type *base);

#if !(defined(FSL_FEATURE_LCDIF_HAS_NO_RESET_PIN) && FSL_FEATURE_LCDIF_HAS_NO_RESET_PIN)

static inline void ELCDIF_PullUpResetPin(LCDIF_Type *base, bool pullUp)
{
    if (pullUp)
    {
        base->CTRL1_SET = LCDIF_CTRL1_RESET_MASK;
    }
    else

```

```

    {
        base->CTRL1_CLR = LCDIF_CTRL1_RESET_MASK;
    }
}
#endif

static inline void ELCDIF_EnablePxpHandShake(LCDIF_Type *base, bool enable)
{
    if (enable)
    {
        base->CTRL_SET = LCDIF_CTRL_ENABLE_PXP_HANDSHAKE_MASK;
    }
    else
    {
        base->CTRL_CLR = LCDIF_CTRL_ENABLE_PXP_HANDSHAKE_MASK;
    }
}

/* @} */

static inline uint32_t ELCDIF_GetCrcValue(LCDIF_Type *base)
{
    return base->CRC_STAT;
}

static inline uint32_t ELCDIF_GetBusMasterErrorAddr(LCDIF_Type *base)
{
    return base->BM_ERROR_STAT;
}

```

## Parameters

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

## Returns

The mask value of status flags, it is OR'ed value of [\\_elcdif\\_status\\_flags](#).

### 17.6.3 static uint32\_t ELCDIF\_GetLfifoCount ( LCDIF\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

The LFIFO current count

### 17.6.4 static void ELCDIF\_EnableInterrupts ( LCDIF\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>mask</i>	interrupt source, OR'ed value of <code>_elcdif_interrupt_enable</code> .

**17.6.5 static void ELCDIF\_DisableInterrupts ( LCDIF\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>mask</i>	interrupt source, OR'ed value of <code>_elcdif_interrupt_enable</code> .

**17.6.6 static uint32\_t ELCDIF\_GetInterruptStatus ( LCDIF\_Type \* *base* )**  
**[inline], [static]**

Parameters

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

Returns

Interrupt pending status, OR'ed value of `_elcdif_interrupt_flags`.

**17.6.7 static void ELCDIF\_ClearInterruptStatus ( LCDIF\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>mask</i>	of the flags to clear, OR'ed value of <code>_elcdif_interrupt_flags</code> .

## Chapter 18

# ENC: Quadrature Encoder/Decoder

### 18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Quadrature Encoder/Decoder (ENC) module of MCUXpresso SDK devices.

This section describes the programming interface of the ENC Peripheral driver. The ENC driver configures the ENC module and provides a functional interface for the user to build the ENC application.

### 18.2 Function groups

#### 18.2.1 Initialization and De-initialization

This function group initializes default configuration structure for the ENC counter and initializes ENC counter with the normal configuration and de-initialize ENC module. Some APIs are also created to control the features.

#### 18.2.2 Status

This function group get/clear the ENC status.

#### 18.2.3 Interrupts

This function group enable/disable the ENC interrupts.

#### 18.2.4 Value Operation

This function group get the counter/hold value of positions.

### 18.3 Typical use case

#### 18.3.1 Polling Configuration

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

### Data Structures

- struct [enc\\_config\\_t](#)

## Typical use case

- *Define user configuration structure for ENC module. [More...](#)*
  - struct `enc_self_test_config_t`
    - *Define configuration structure for self test module. [More...](#)*

## Macros

- #define `FSL_ENC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## Enumerations

- enum `_enc_interrupt_enable` {  
    `kENC_HOMETransitionInterruptEnable` = (1U << 0U),  
    `kENC_INDEXPulseInterruptEnable` = (1U << 1U),  
    `kENC_WatchdogTimeoutInterruptEnable` = (1U << 2U),  
    `kENC_PositionCompareInterruptEnable` = (1U << 3U),  
    `kENC_SimultBothPhaseChangeInterruptEnable`,  
    `kENC_PositionRollOverInterruptEnable` = (1U << 5U),  
    `kENC_PositionRollUnderInterruptEnable` = (1U << 6U) }  
*Interrupt enable/disable mask.*
- enum `_enc_status_flags` {  
    `kENC_HOMETransitionFlag` = (1U << 0U),  
    `kENC_INDEXPulseFlag` = (1U << 1U),  
    `kENC_WatchdogTimeoutFlag` = (1U << 2U),  
    `kENC_PositionCompareFlag` = (1U << 3U),  
    `kENC_SimultBothPhaseChangeFlag` = (1U << 4U),  
    `kENC_PositionRollOverFlag` = (1U << 5U),  
    `kENC_PositionRollUnderFlag` = (1U << 6U),  
    `kENC_LastCountDirectionFlag` = (1U << 7U) }  
*Status flag mask.*
- enum `_enc_signal_status_flags` {  
    `kENC_RawHOMESTatusFlag` = ENC\_IMR\_HOME\_MASK,  
    `kENC_RawINDEXStatusFlag` = ENC\_IMR\_INDEX\_MASK,  
    `kENC_RawPHBStatusFlag` = ENC\_IMR\_PHB\_MASK,  
    `kENC_RawPHAEXStatusFlag` = ENC\_IMR\_PHA\_MASK,  
    `kENC_FilteredHOMESTatusFlag` = ENC\_IMR\_FHOM\_MASK,  
    `kENC_FilteredINDEXStatusFlag` = ENC\_IMR\_FIND\_MASK,  
    `kENC_FilteredPHBStatusFlag` = ENC\_IMR\_FPHB\_MASK,  
    `kENC_FilteredPHAStatusFlag` = ENC\_IMR\_FPHA\_MASK }  
*Signal status flag mask.*
- enum `enc_home_trigger_mode_t` {  
    `kENC_HOMETriggerDisabled` = 0U,  
    `kENC_HOMETriggerOnRisingEdge`,  
    `kENC_HOMETriggerOnFallingEdge` }  
*Define HOME signal's trigger mode.*
- enum `enc_index_trigger_mode_t` {



- ```
kENC_INDEXTriggerDisabled = 0U,
kENC_INDEXTriggerOnRisingEdge,
kENC_INDEXTriggerOnFallingEdge }
    Define INDEX signal's trigger mode.
```
- enum `enc_decoder_work_mode_t` {

```
kENC_DecoderWorkAsNormalMode = 0U,
kENC_DecoderWorkAsSignalPhaseCountMode }
```

 Define type for decoder work mode.
  - enum `enc_position_match_mode_t` {

```
kENC_POSMATCHOnPositionCounterEqualToComapreValue = 0U,
kENC_POSMATCHOnReadingAnyPositionCounter }
```

 Define type for the condition of POSMATCH pulses.
  - enum `enc_revolution_count_condition_t` {

```
kENC_RevolutionCountOnINDEXPulse = 0U,
kENC_RevolutionCountOnRollOverModulus }
```

 Define type for determining how the revolution counter (REV) is incremented/decremented.
  - enum `enc_self_test_direction_t` {

```
kENC_SelfTestDirectionPositive = 0U,
kENC_SelfTestDirectionNegative }
```

 Define type for direction of self test generated signal.

## Variables

- bool `enc_config_t::enableReverseDirection`  
Enable reverse direction counting.
- `enc_decoder_work_mode_t enc_config_t::decoderWorkMode`  
Enable signal phase count mode.
- `enc_home_trigger_mode_t enc_config_t::HOMETriggerMode`  
Enable HOME to initialize position counters.
- `enc_index_trigger_mode_t enc_config_t::INDEXTriggerMode`  
Enable INDEX to initialize position counters.
- bool `enc_config_t::enableTRIGGERClearPositionCounter`  
Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.
- bool `enc_config_t::enableTRIGGERClearHoldPositionCounter`  
Enable update of hold registers on rising edge of TRIGGER, or not.
- bool `enc_config_t::enableWatchdog`  
Enable the watchdog to detect if the target is moving or not.
- uint16\_t `enc_config_t::watchdogTimeoutValue`  
Watchdog timeout count value.
- uint16\_t `enc_config_t::filterCount`  
Input Filter Sample Count.
- uint16\_t `enc_config_t::filterSamplePeriod`  
Input Filter Sample Period.
- `enc_position_match_mode_t enc_config_t::positionMatchMode`  
The condition of POSMATCH pulses.
- uint32\_t `enc_config_t::positionCompareValue`  
Position compare value.
- `enc_revolution_count_condition_t enc_config_t::revolutionCountCondition`  
Revolution Counter Modulus Enable.
- bool `enc_config_t::enableModuloCountMode`

## Typical use case

- *Enable Modulo Counting.*
- `uint32_t enc_config_t::positionModulusValue`  
*Position modulus value.*
- `uint32_t enc_config_t::positionInitialValue`  
*Position initial value.*
- `enc_self_test_direction_t enc_self_test_config_t::signalDirection`  
*Direction of self test generated signal.*
- `uint16_t enc_self_test_config_t::signalCount`  
*Hold the number of quadrature advances to generate.*
- `uint16_t enc_self_test_config_t::signalPeriod`  
*Hold the period of quadrature phase in IPBus clock cycles.*

## Initialization and De-initialization

- `void ENC_Init (ENC_Type *base, const enc_config_t *config)`  
*Initialization for the ENC module.*
- `void ENC_Deinit (ENC_Type *base)`  
*De-initialization for the ENC module.*
- `void ENC_GetDefaultConfig (enc_config_t *config)`  
*Get an available pre-defined settings for ENC's configuration.*
- `void ENC_DoSoftwareLoadInitialPositionValue (ENC_Type *base)`  
*Load the initial position value to position counter.*
- `void ENC_SetSelfTestConfig (ENC_Type *base, const enc_self_test_config_t *config)`  
*Enable and configure the self test function.*
- `void ENC_EnableWatchdog (ENC_Type *base, bool enable)`  
*Enable watchdog for ENC module.*
- `void ENC_SetInitialPositionValue (ENC_Type *base, uint32_t value)`  
*Set initial position value for ENC module.*

## Status

- `uint32_t ENC_GetStatusFlags (ENC_Type *base)`  
*Get the status flags.*
- `void ENC_ClearStatusFlags (ENC_Type *base, uint32_t mask)`  
*Clear the status flags.*
- `static uint16_t ENC_GetSignalStatusFlags (ENC_Type *base)`  
*Get the signals' real-time status.*

## Interrupts

- `void ENC_EnableInterrupts (ENC_Type *base, uint32_t mask)`  
*Enable the interrupts.*
- `void ENC_DisableInterrupts (ENC_Type *base, uint32_t mask)`  
*Disable the interrupts.*
- `uint32_t ENC_GetEnabledInterrupts (ENC_Type *base)`  
*Get the enabled interrupts' flags.*

## Value Operation

- `uint32_t ENC_GetPositionValue (ENC_Type *base)`  
*Get the current position counter's value.*

- uint32\_t [ENC\\_GetHoldPositionValue](#) (ENC\_Type \*base)  
*Get the hold position counter's value.*
- static uint16\_t [ENC\\_GetPositionDifferenceValue](#) (ENC\_Type \*base)  
*Get the position difference counter's value.*
- static uint16\_t [ENC\\_GetHoldPositionDifferenceValue](#) (ENC\_Type \*base)  
*Get the hold position difference counter's value.*
- static uint16\_t [ENC\\_GetRevolutionValue](#) (ENC\_Type \*base)  
*Get the position revolution counter's value.*
- static uint16\_t [ENC\\_GetHoldRevolutionValue](#) (ENC\_Type \*base)  
*Get the hold position revolution counter's value.*

## 18.4 Data Structure Documentation

### 18.4.1 struct enc\_config\_t

#### Data Fields

- bool [enableReverseDirection](#)  
*Enable reverse direction counting.*
- [enc\\_decoder\\_work\\_mode\\_t](#) [decoderWorkMode](#)  
*Enable signal phase count mode.*
- [enc\\_home\\_trigger\\_mode\\_t](#) [HOMETriggerMode](#)  
*Enable HOME to initialize position counters.*
- [enc\\_index\\_trigger\\_mode\\_t](#) [INDEXTriggerMode](#)  
*Enable INDEX to initialize position counters.*
- bool [enableTRIGGERClearPositionCounter](#)  
*Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.*
- bool [enableTRIGGERClearHoldPositionCounter](#)  
*Enable update of hold registers on rising edge of TRIGGER, or not.*
- bool [enableWatchdog](#)  
*Enable the watchdog to detect if the target is moving or not.*
- uint16\_t [watchdogTimeoutValue](#)  
*Watchdog timeout count value.*
- uint16\_t [filterCount](#)  
*Input Filter Sample Count.*
- uint16\_t [filterSamplePeriod](#)  
*Input Filter Sample Period.*
- [enc\\_position\\_match\\_mode\\_t](#) [positionMatchMode](#)  
*The condition of POSMATCH pulses.*
- uint32\_t [positionCompareValue](#)  
*Position compare value.*
- [enc\\_revolution\\_count\\_condition\\_t](#) [revolutionCountCondition](#)  
*Revolution Counter Modulus Enable.*
- bool [enableModuloCountMode](#)  
*Enable Modulo Counting.*
- uint32\_t [positionModulusValue](#)  
*Position modulus value.*
- uint32\_t [positionInitialValue](#)  
*Position initial value.*

## Enumeration Type Documentation

### 18.4.2 struct enc\_self\_test\_config\_t

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

#### Data Fields

- [enc\\_self\\_test\\_direction\\_t](#) `signalDirection`  
*Direction of self test generated signal.*
- [uint16\\_t](#) `signalCount`  
*Hold the number of quadrature advances to generate.*
- [uint16\\_t](#) `signalPeriod`  
*Hold the period of quadrature phase in IPBus clock cycles.*

## 18.5 Macro Definition Documentation

### 18.5.1 #define FSL\_ENC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 18.6 Enumeration Type Documentation

### 18.6.1 enum \_enc\_interrupt\_enable

Enumerator

- kENC\_HOMETransitionInterruptEnable* HOME interrupt enable.
- kENC\_INDEXPulseInterruptEnable* INDEX pulse interrupt enable.
- kENC\_WatchdogTimeoutInterruptEnable* Watchdog timeout interrupt enable.
- kENC\_PositionCompareInterruptEnable* Position compare interrupt enable.
- kENC\_SimultBothPhaseChangeInterruptEnable* Simultaneous PHASEA and PHASEB change interrupt enable.
- kENC\_PositionRollOverInterruptEnable* Roll-over interrupt enable.
- kENC\_PositionRollUnderInterruptEnable* Roll-under interrupt enable.

### 18.6.2 enum \_enc\_status\_flags

These flags indicate the counter's events.

Enumerator

- kENC\_HOMETransitionFlag* HOME signal transition interrupt request.
- kENC\_INDEXPulseFlag* INDEX Pulse Interrupt Request.
- kENC\_WatchdogTimeoutFlag* Watchdog timeout interrupt request.
- kENC\_PositionCompareFlag* Position compare interrupt request.
- kENC\_SimultBothPhaseChangeFlag* Simultaneous PHASEA and PHASEB change interrupt request.

- kENC\_PositionRollOverFlag* Roll-over interrupt request.
- kENC\_PositionRollUnderFlag* Roll-under interrupt request.
- kENC\_LastCountDirectionFlag* Last count was in the up direction, or the down direction.

### 18.6.3 enum\_enc\_signal\_status\_flags

These flags indicate the counter's signal.

Enumerator

- kENC\_RawHOMESTatusFlag* Raw HOME input.
- kENC\_RawINDEXStatusFlag* Raw INDEX input.
- kENC\_RawPHBStatusFlag* Raw PHASEB input.
- kENC\_RawPHAEXStatusFlag* Raw PHASEA input.
- kENC\_FilteredHOMESTatusFlag* The filtered version of HOME input.
- kENC\_FilteredINDEXStatusFlag* The filtered version of INDEX input.
- kENC\_FilteredPHBStatusFlag* The filtered version of PHASEB input.
- kENC\_FilteredPHAStatusFlag* The filtered version of PHASEA input.

### 18.6.4 enum\_enc\_home\_trigger\_mode\_t

The ENC would count the trigger from HOME signal line.

Enumerator

- kENC\_HOMETriggerDisabled* HOME signal's trigger is disabled.
- kENC\_HOMETriggerOnRisingEdge* Use positive going edge-to-trigger initialization of position counters.
- kENC\_HOMETriggerOnFallingEdge* Use negative going edge-to-trigger initialization of position counters.

### 18.6.5 enum\_enc\_index\_trigger\_mode\_t

The ENC would count the trigger from INDEX signal line.

Enumerator

- kENC\_INDEXTriggerDisabled* INDEX signal's trigger is disabled.
- kENC\_INDEXTriggerOnRisingEdge* Use positive going edge-to-trigger initialization of position counters.
- kENC\_INDEXTriggerOnFallingEdge* Use negative going edge-to-trigger initialization of position counters.

## Enumeration Type Documentation

### 18.6.6 enum enc\_decoder\_work\_mode\_t

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

Enumerator

***kENC\_DecoderWorkAsNormalMode*** Use standard quadrature decoder with PHASEA and PHASEB.

***kENC\_DecoderWorkAsSignalPhaseCountMode*** PHASEA input generates a count signal while PHASEB input control the direction.

### 18.6.7 enum enc\_position\_match\_mode\_t

Enumerator

***kENC\_POSMATCHOnPositionCounterEqualToCompareValue*** POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

***kENC\_POSMATCHOnReadingAnyPositionCounter*** POSMATCH pulses when any position counter register is read.

### 18.6.8 enum enc\_revolution\_count\_condition\_t

Enumerator

***kENC\_RevolutionCountOnINDEXPulse*** Use INDEX pulse to increment/decrement revolution counter.

***kENC\_RevolutionCountOnRollOverModulus*** Use modulus counting roll-over/under to increment/decrement revolution counter.

### 18.6.9 enum enc\_self\_test\_direction\_t

Enumerator

***kENC\_SelfTestDirectionPositive*** Self test generates the signal in positive direction.

***kENC\_SelfTestDirectionNegative*** Self test generates the signal in negative direction.

## 18.7 Function Documentation

### 18.7.1 void ENC\_Init ( ENC\_Type \* *base*, const enc\_config\_t \* *config* )

This function is to make the initialization for the ENC module. It should be called firstly before any operation to the ENC with the operations like:

- Enable the clock for ENC module.
- Configure the ENC's working attributes.

Parameters

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

### 18.7.2 void ENC\_Deinit ( ENC\_Type \* *base* )

This function is to make the de-initialization for the ENC module. It could be called when ENC is no longer used with the operations like:

- Disable the clock for ENC module.

Parameters

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

### 18.7.3 void ENC\_GetDefaultConfig ( enc\_config\_t \* *config* )

This function initializes the ENC configuration structure with an available settings, the default value are:

```

* config->enableReverseDirection      = false;
* config->decoderWorkMode              = kENC_DecoderWorkAsNormalMode
  ;
* config->HOMETriggerMode              = kENC_HOMETriggerDisabled;
* config->INDEXTriggerMode             = kENC_INDEXTriggerDisabled;
* config->enableTRIGGERClearPositionCounter = false;
* config->enableTRIGGERClearHoldPositionCounter = false;
* config->enableWatchdog                = false;
* config->watchdogTimeoutValue          = 0U;
* config->filterCount                   = 0U;
* config->filterSamplePeriod            = 0U;
* config->positionMatchMode             =
  kENC_POSMATCHOnPositionCounterEqualToComapreValue;
* config->positionCompareValue          = 0xFFFFFFFFFU;
* config->revolutionCountCondition      =
  kENC_RevolutionCountOnINDEXPulse;
* config->enableModuloCountMode         = false;
* config->positionModulusValue          = 0U;
* config->positionInitialValue          = 0U;
*

```

## Function Documentation

### Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>config</i> | Pointer to a variable of configuration structure. See to "enc_config_t". |
|---------------|--------------------------------------------------------------------------|

### 18.7.4 void ENC\_DoSoftwareLoadInitialPositionValue ( ENC\_Type \* *base* )

This function is to transfer the initial position value (UNIT and LINIT) contents to position counter (UP-OS and LPOS), so that to provide the consistent operation the position counter registers.

### Parameters

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

### 18.7.5 void ENC\_SetSelfTestConfig ( ENC\_Type \* *base*, const enc\_self\_test\_config\_t \* *config* )

This function is to enable and configuration the self test function. It controls and sets the frequency of a quadrature signal generator. It provides a quadrature test signal to the inputs of the quadrature decoder module. It is a factory test feature; however, it may be useful to customers' software development and testing.

### Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | ENC peripheral base address.                                                                 |
| <i>config</i> | Pointer to configuration structure. See to "enc_self_test_config_t". Pass "NULL" to disable. |

### 18.7.6 void ENC\_EnableWatchdog ( ENC\_Type \* *base*, bool *enable* )

### Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | ENC peripheral base address      |
| <i>enable</i> | Enables or disables the watchdog |

### 18.7.7 void ENC\_SetInitialPositionValue ( ENC\_Type \* *base*, uint32\_t *value* )



Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | ENC peripheral base address |
| <i>value</i> | Positive initial value      |

### 18.7.8 uint32\_t ENC\_GetStatusFlags ( ENC\_Type \* *base* )

Parameters

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

Returns

Mask value of status flags. For available mask, see to "\_enc\_status\_flags".

### 18.7.9 void ENC\_ClearStatusFlags ( ENC\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                           |
|-------------|-------------------------------------------------------------------------------------------|
| <i>base</i> | ENC peripheral base address.                                                              |
| <i>mask</i> | Mask value of status flags to be cleared. For available mask, see to "_enc_status_flags". |

### 18.7.10 static uint16\_t ENC\_GetSignalStatusFlags ( ENC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Mask value of signals' real-time status. For available mask, see to "\_enc\_signal\_status\_flags"

### 18.7.11 void ENC\_EnableInterrupts ( ENC\_Type \* *base*, uint32\_t *mask* )

## Function Documentation

### Parameters

|             |                                                                                             |
|-------------|---------------------------------------------------------------------------------------------|
| <i>base</i> | ENC peripheral base address.                                                                |
| <i>mask</i> | Mask value of interrupts to be enabled. For available mask, see to "_enc_interrupt_enable". |

### 18.7.12 void ENC\_DisableInterrupts ( ENC\_Type \* *base*, uint32\_t *mask* )

### Parameters

|             |                                                                                              |
|-------------|----------------------------------------------------------------------------------------------|
| <i>base</i> | ENC peripheral base address.                                                                 |
| <i>mask</i> | Mask value of interrupts to be disabled. For available mask, see to "_enc_interrupt_enable". |

### 18.7.13 uint32\_t ENC\_GetEnabledInterrupts ( ENC\_Type \* *base* )

### Parameters

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

### Returns

Mask value of enabled interrupts.

### 18.7.14 uint32\_t ENC\_GetPositionValue ( ENC\_Type \* *base* )

### Parameters

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

### Returns

Current position counter's value.

**18.7.15 uint32\_t ENC\_GetHoldPositionValue ( ENC\_Type \* *base* )**

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

## Function Documentation

### Parameters

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

### Returns

Hold position counter's value.

**18.7.16 static uint16\_t ENC\_GetPositionDifferenceValue ( ENC\_Type \* *base* )  
[inline], [static]**

### Parameters

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

### Returns

The position difference counter's value.

**18.7.17 static uint16\_t ENC\_GetHoldPositionDifferenceValue ( ENC\_Type \* *base* )  
[inline], [static]**

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

### Parameters

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

### Returns

Hold position difference counter's value.

**18.7.18 static uint16\_t ENC\_GetRevolutionValue ( ENC\_Type \* *base* ) [inline],  
[static]**

## Parameters

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

## Returns

The position revolution counter's value.

### 18.7.19 `static uint16_t ENC_GetHoldRevolutionValue ( ENC_Type * base )` `[inline], [static]`

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

## Parameters

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

## Returns

Hold position revolution counter's value.

## 18.8 Variable Documentation

### 18.8.1 `bool enc_config_t::enableReverseDirection`

### 18.8.2 `enc_decoder_work_mode_t enc_config_t::decoderWorkMode`

### 18.8.3 `enc_home_trigger_mode_t enc_config_t::HOMETriggerMode`

### 18.8.4 `enc_index_trigger_mode_t enc_config_t::INDEXTriggerMode`

### 18.8.5 `bool enc_config_t::enableTRIGGERClearPositionCounter`

### 18.8.6 `bool enc_config_t::enableWatchdog`

### 18.8.7 `uint16_t enc_config_t::watchdogTimeoutValue`

It stores the timeout count for the quadrature decoder module watchdog timer. This field is only available when "enableWatchdog" = true. The available value is a 16-bit unsigned number.

## Variable Documentation

### 18.8.8 uint16\_t enc\_config\_t::filterCount

This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. The Available range is 0 - 7.

### 18.8.9 uint16\_t enc\_config\_t::filterSamplePeriod

This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available range is 0 - 255.

### 18.8.10 enc\_position\_match\_mode\_t enc\_config\_t::positionMatchMode

### 18.8.11 uint32\_t enc\_config\_t::positionCompareValue

The available value is a 32-bit number.

### 18.8.12 enc\_revolution\_count\_condition\_t enc\_config\_t::revolutionCountCondition

### 18.8.13 bool enc\_config\_t::enableModuloCountMode

### 18.8.14 uint32\_t enc\_config\_t::positionModulusValue

This value would be available only when "enableModuloCountMode" = true. The available value is a 32-bit number.

### 18.8.15 uint32\_t enc\_config\_t::positionInitialValue

The available value is a 32-bit number.

### 18.8.16 enc\_self\_test\_direction\_t enc\_self\_test\_config\_t::signalDirection

### 18.8.17 uint16\_t enc\_self\_test\_config\_t::signalCount

The available range is 0 - 255.

### 18.8.18 uint16\_t enc\_self\_test\_config\_t::signalPeriod

The available range is 0 - 31.





# Chapter 19

## ENET: Ethernet MAC Driver

### 19.1 Overview

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

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

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

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

1. For single ring For ENET receive, the [ENET\\_GetRxFrameSize\(\)](#) function needs to be called to get the received data size. Then, call the [ENET\\_ReadFrame\(\)](#) function to get the received data. If the received error occurs, call the [ENET\\_GetRxErrBeforeReadFrame\(\)](#) function after [ENET\\_GetRxFrameSize\(\)](#) and before [ENET\\_ReadFrame\(\)](#) functions to get the detailed error information.

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

1. For multiple-ring supported The ENET driver now added a series transactional APIs with postfix "-MultiRing" to support the extended multiple-ring for AVB feature. There are extended multiple-ring functions for receive side: [ENET\\_GetRxErrBeforeReadFrameMultiRing\(\)](#), [ENET\\_GetRxFrameSizeMultiRing\(\)](#), and [ENET\\_ReadFrameMultiRing\(\)](#). They are the similar to the single ring receive APIs and only add the "ringId" input param to identify the different ring index. For TX side add the [ENET\\_SendFrameMultiRing\(\)](#), [ENET\\_GetTxErrAfterSendFrameMultiRing\(\)](#). They are the similar to the single ring transmit APIs and only add the "ringId" input param to identify the different ring

## Typical use case

index.

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

The [ENET\\_Ptp1588Configure\(\)](#) function needs to be called when the ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE is defined and the IEEE 1588 feature is required. The [ENET\\_GetRxFrameTime\(\)](#) and [ENET\\_GetTxFrameTime\(\)](#) functions are called by the PTP stack to get the timestamp captured by the ENET driver.

## 19.2 Typical use case

### 19.2.1 ENET Initialization, receive, and transmit operations

For the ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE undefined use case, use the legacy type buffer descriptor transmit/receive the frame as follows. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/enet For the ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE defined use case, add the PTP IEEE 1588 configuration to enable the PTP IEEE 1588 feature. The initialization occurs as follows. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/enet

## Data Structures

- struct [enet\\_rx\\_bd\\_struct\\_t](#)  
*Defines the receive buffer descriptor structure for the little endian system. [More...](#)*
- struct [enet\\_tx\\_bd\\_struct\\_t](#)  
*Defines the enhanced transmit buffer descriptor structure for the little endian system. [More...](#)*
- struct [enet\\_data\\_error\\_stats\\_t](#)  
*Defines the ENET data error statistic structure. [More...](#)*
- struct [enet\\_buffer\\_config\\_t](#)  
*Defines the receive buffer descriptor configuration structure. [More...](#)*
- struct [enet\\_ptp\\_time\\_t](#)  
*Defines the ENET PTP time stamp structure. [More...](#)*
- struct [enet\\_ptp\\_time\\_data\\_t](#)  
*Defines the structure for the ENET PTP message data and timestamp data. [More...](#)*
- struct [enet\\_ptp\\_time\\_data\\_ring\\_t](#)  
*Defines the ENET PTP ring buffer structure for the PTP message timestamp store. [More...](#)*
- struct [enet\\_ptp\\_config\\_t](#)  
*Defines the ENET PTP configuration structure. [More...](#)*
- struct [enet\\_intcoalesce\\_config\\_t](#)  
*Defines the interrupt coalescing configure structure. [More...](#)*
- struct [enet\\_config\\_t](#)  
*Defines the basic configuration structure for the ENET device. [More...](#)*
- struct [enet\\_handle\\_t](#)  
*Defines the ENET handler structure. [More...](#)*

## Macros

- #define [ENET\\_BUFFDESCRIPTOR\\_RX\\_ERR\\_MASK](#)  
*Defines the receive error status flag mask.*

## Typedefs

- typedef void(\* [enet\\_callback\\_t](#))(ENET\_Type \*base, enet\_handle\_t \*handle, [enet\\_event\\_t](#) event, void \*userData)  
*ENET callback function.*

## Enumerations

- enum [\\_enet\\_status](#) {  
[kStatus\\_ENET\\_RxFrameError](#) = MAKE\_STATUS(kStatusGroup\_ENET, 0U),  
[kStatus\\_ENET\\_RxFrameFail](#) = MAKE\_STATUS(kStatusGroup\_ENET, 1U),  
[kStatus\\_ENET\\_RxFrameEmpty](#) = MAKE\_STATUS(kStatusGroup\_ENET, 2U),  
[kStatus\\_ENET\\_TxFrameOverLen](#) = MAKE\_STATUS(kStatusGroup\_ENET, 3U),  
[kStatus\\_ENET\\_TxFrameBusy](#) = MAKE\_STATUS(kStatusGroup\_ENET, 4U),  
[kStatus\\_ENET\\_TxFrameFail](#) = MAKE\_STATUS(kStatusGroup\_ENET, 5U),  
[kStatus\\_ENET\\_PtpTsRingFull](#) = MAKE\_STATUS(kStatusGroup\_ENET, 6U),  
[kStatus\\_ENET\\_PtpTsRingEmpty](#) = MAKE\_STATUS(kStatusGroup\_ENET, 7U) }  
*Defines the status return codes for transaction.*
- enum [enet\\_mii\\_mode\\_t](#) {  
[kENET\\_MiiMode](#) = 0U,  
[kENET\\_RmiiMode](#) = 1U }  
*Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.*
- enum [enet\\_mii\\_speed\\_t](#) {  
[kENET\\_MiiSpeed10M](#) = 0U,  
[kENET\\_MiiSpeed100M](#) = 1U }  
*Defines the 10/100/1000 Mbps speed for the MII data interface.*
- enum [enet\\_mii\\_duplex\\_t](#) {  
[kENET\\_MiiHalfDuplex](#) = 0U,  
[kENET\\_MiiFullDuplex](#) }  
*Defines the half or full duplex for the MII data interface.*
- enum [enet\\_mii\\_write\\_t](#) {  
[kENET\\_MiiWriteNoCompliant](#) = 0U,  
[kENET\\_MiiWriteValidFrame](#) }  
*Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.*
- enum [enet\\_mii\\_read\\_t](#) {  
[kENET\\_MiiReadValidFrame](#) = 2U,  
[kENET\\_MiiReadNoCompliant](#) = 3U }  
*Defines the read operation for the MII management frame.*
- enum [enet\\_mii\\_extend\\_opcode](#) {  
[kENET\\_MiiAddrWrite\\_C45](#) = 0U,  
[kENET\\_MiiWriteFrame\\_C45](#) = 1U,  
[kENET\\_MiiReadFrame\\_C45](#) = 3U }  
*Define the MII opcode for extended MDIO\_CLAUSES\_45 Frame.*
- enum [enet\\_special\\_control\\_flag\\_t](#) {

## Typical use case

```
kENET_ControlFlowControlEnable = 0x0001U,  
kENET_ControlRxPayloadCheckEnable = 0x0002U,  
kENET_ControlRxPadRemoveEnable = 0x0004U,  
kENET_ControlRxBroadCastRejectEnable = 0x0008U,  
kENET_ControlMacAddrInsert = 0x0010U,  
kENET_ControlStoreAndFwdDisable = 0x0020U,  
kENET_ControlSMIPreambleDisable = 0x0040U,  
kENET_ControlPromiscuousEnable = 0x0080U,  
kENET_ControlMIILoopEnable = 0x0100U,  
kENET_ControlVLANTagEnable = 0x0200U }
```

*Defines a special configuration for ENET MAC controller.*

- enum `enet_interrupt_enable_t` {  
kENET\_BabrInterrupt = ENET\_EIR\_BABR\_MASK,  
kENET\_BabtInterrupt = ENET\_EIR\_BABT\_MASK,  
kENET\_GraceStopInterrupt = ENET\_EIR\_GRA\_MASK,  
kENET\_TxFrameInterrupt = ENET\_EIR\_TXF\_MASK,  
kENET\_TxBufferInterrupt = ENET\_EIR\_TXB\_MASK,  
kENET\_RxFrameInterrupt = ENET\_EIR\_RXF\_MASK,  
kENET\_RxBufferInterrupt = ENET\_EIR\_RXB\_MASK,  
kENET\_MiiInterrupt = ENET\_EIR\_MII\_MASK,  
kENET\_EBusERInterrupt = ENET\_EIR\_EBERR\_MASK,  
kENET\_LateCollisionInterrupt = ENET\_EIR\_LC\_MASK,  
kENET\_RetryLimitInterrupt = ENET\_EIR\_RL\_MASK,  
kENET\_UnderrunInterrupt = ENET\_EIR\_UN\_MASK,  
kENET\_PayloadRxInterrupt = ENET\_EIR\_PLR\_MASK,  
kENET\_WakeupInterrupt = ENET\_EIR\_WAKEUP\_MASK,  
kENET\_TsAvailInterrupt = ENET\_EIR\_TS\_AVAIL\_MASK,  
kENET\_TsTimerInterrupt = ENET\_EIR\_TS\_TIMER\_MASK }

*List of interrupts supported by the peripheral.*

- enum `enet_event_t` {  
kENET\_RxEvent,  
kENET\_TxEvent,  
kENET\_ErrEvent,  
kENET\_WakeUpEvent,  
kENET\_TimeStampEvent,  
kENET\_TimeStampAvailEvent }

*Defines the common interrupt event for callback use.*

- enum `enet_tx_accelerator_t` {  
kENET\_TxAccelIsShift16Enabled = ENET\_TACC\_SHIFT16\_MASK,  
kENET\_TxAccelIpCheckEnabled = ENET\_TACC\_IPCHK\_MASK,  
kENET\_TxAccelProtoCheckEnabled = ENET\_TACC\_PROCHK\_MASK }

*Defines the transmit accelerator configuration.*

- enum `enet_rx_accelerator_t` {

```

kENET_RxAccelPadRemoveEnabled = ENET_RACC_PADREM_MASK,
kENET_RxAccelIpCheckEnabled = ENET_RACC_IPDIS_MASK,
kENET_RxAccelProtoCheckEnabled = ENET_RACC_PRODIS_MASK,
kENET_RxAccelMacCheckEnabled = ENET_RACC_LINEDIS_MASK,
kENET_RxAccelIsShift16Enabled = ENET_RACC_SHIFT16_MASK }

```

*Defines the receive accelerator configuration.*

- enum `enet_ptp_event_type_t` {
 

```

kENET_PtpEventMsgType = 3U,
kENET_PtpSrcPortIdLen = 10U,
kENET_PtpEventPort = 319U,
kENET_PtpGnrlPort = 320U }

```

*Defines the ENET PTP message related constant.*

- enum `enet_ptp_timer_channel_t` {
 

```

kENET_PtpTimerChannel1 = 0U,
kENET_PtpTimerChannel2,
kENET_PtpTimerChannel3,
kENET_PtpTimerChannel4 }

```

*Defines the IEEE 1588 PTP timer channel numbers.*

- enum `enet_ptp_timer_channel_mode_t` {
 

```

kENET_PtpChannelDisable = 0U,
kENET_PtpChannelRisingCapture = 1U,
kENET_PtpChannelFallingCapture = 2U,
kENET_PtpChannelBothCapture = 3U,
kENET_PtpChannelSoftCompare = 4U,
kENET_PtpChannelToggleCompare = 5U,
kENET_PtpChannelClearCompare = 6U,
kENET_PtpChannelSetCompare = 7U,
kENET_PtpChannelClearCompareSetOverflow = 10U,
kENET_PtpChannelSetCompareClearOverflow = 11U,
kENET_PtpChannelPulseLowonCompare = 14U,
kENET_PtpChannelPulseHighonCompare = 15U }

```

*Defines the capture or compare mode for IEEE 1588 PTP timer channels.*

## Driver version

- #define `FSL_ENET_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 3))
 

*Defines the driver version.*

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

Defines the queue number.

- #define `ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK` 0x8000U
 

*Empty bit mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK` 0x4000U
 

*Software owner one mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_WRAP_MASK` 0x2000U
 

*Next buffer descriptor is the start address.*

## Typical use case

- #define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER2\_Mask 0x1000U  
*Software owner two mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_LAST\_MASK 0x0800U  
*Last BD of the frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_MISS\_MASK 0x0100U  
*Received because of the promiscuous mode.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_BROADCAST\_MASK 0x0080U  
*Broadcast packet mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_MULTICAST\_MASK 0x0040U  
*Multicast packet mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_LENVIOLATE\_MASK 0x0020U  
*Length violation mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_NOOCTET\_MASK 0x0010U  
*Non-octet aligned frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_CRC\_MASK 0x0004U  
*CRC error mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_OVERRUN\_MASK 0x0002U  
*FIFO overrun mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_TRUNC\_MASK 0x0001U  
*Frame is truncated mask.*

## Control and status bit masks of the transmit buffer descriptor.

- #define ENET\_BUFFDESCRIPTOR\_TX\_READY\_MASK 0x8000U  
*Ready bit mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWNER1\_MASK 0x4000U  
*Software owner one mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_WRAP\_MASK 0x2000U  
*Wrap buffer descriptor mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWNER2\_MASK 0x1000U  
*Software owner two mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_LAST\_MASK 0x0800U  
*Last BD of the frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_TRANSMITCRC\_MASK 0x0400U  
*Transmit CRC mask.*

## First extended control region bit masks of the receive buffer descriptor.

- #define ENET\_BUFFDESCRIPTOR\_RX\_IPV4\_MASK 0x0001U  
*Ipv4 frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_IPV6\_MASK 0x0002U  
*Ipv6 frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_VLAN\_MASK 0x0004U  
*VLAN frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_PROTOCOLCHECKSUM\_MASK 0x0010U  
*Protocol checksum error mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_IPHEADCHECKSUM\_MASK 0x0020U  
*IP header checksum error mask.*

**Second extended control region bit masks of the receive buffer descriptor.**

- #define ENET\_BUFFDESCRIPTOR\_RX\_INTERRUPT\_MASK 0x0080U  
*BD interrupt mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_UNICAST\_MASK 0x0100U  
*Unicast frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_COLLISION\_MASK 0x0200U  
*BD collision mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_PHYERR\_MASK 0x0400U  
*PHY error mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_MACERR\_MASK 0x8000U  
*Mac error mask.*

**First extended control region bit masks of the transmit buffer descriptor.**

- #define ENET\_BUFFDESCRIPTOR\_TX\_ERR\_MASK 0x8000U  
*Transmit error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_UNDERFLOWERR\_MASK 0x2000U  
*Underflow error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_EXCCOLLISIONERR\_MASK 0x1000U  
*Excess collision error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_FRAMEERR\_MASK 0x0800U  
*Frame error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_LATECOLLISIONERR\_MASK 0x0400U  
*Late collision error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_OVERFLOWERR\_MASK 0x0200U  
*Overflow error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_TIMESTAMPERR\_MASK 0x0100U  
*Timestamp error mask.*

**Second extended control region bit masks of the transmit buffer descriptor.**

- #define ENET\_BUFFDESCRIPTOR\_TX\_INTERRUPT\_MASK 0x4000U  
*Interrupt mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_TIMESTAMP\_MASK 0x2000U  
*Timestamp flag mask.*

**Defines some Ethernet parameters.**

- #define ENET\_FRAME\_MAX\_FRAMELEN 1518U  
*Default maximum Ethernet frame size.*
- #define ENET\_FIFO\_MIN\_RX\_FULL 5U  
*ENET minimum receive FIFO full.*
- #define ENET\_RX\_MIN\_BUFFERSIZE 256U  
*ENET minimum buffer size.*
- #define ENET\_PHY\_MAXADDRESS (ENET\_MMFR\_PA\_MASK >> ENET\_MMFR\_PA\_SHIFT)
- #define ENET\_TX\_INTERRUPT (kENET\_TxFrameInterrupt | kENET\_TxBufferInterrupt)
- #define ENET\_RX\_INTERRUPT (kENET\_RxFrameInterrupt | kENET\_RxBufferInterrupt)
- #define ENET\_TS\_INTERRUPT (kENET\_TsTimerInterrupt | kENET\_TsAvailInterrupt)
- #define ENET\_ERR\_INTERRUPT
- #define ENET\_ERR\_INTERRUPT

## Typical use case

### Initialization and De-initialization

- void [ENET\\_GetDefaultConfig](#) ([enet\\_config\\_t](#) \*config)  
*Gets the ENET default configuration structure.*
- void [ENET\\_Init](#) ([ENET\\_Type](#) \*base, [enet\\_handle\\_t](#) \*handle, const [enet\\_config\\_t](#) \*config, const [enet\\_buffer\\_config\\_t](#) \*bufferConfig, [uint8\\_t](#) \*macAddr, [uint32\\_t](#) srcClock\_Hz)  
*Initializes the ENET module.*
- void [ENET\\_Deinit](#) ([ENET\\_Type](#) \*base)  
*Deinitializes the ENET module.*
- static void [ENET\\_Reset](#) ([ENET\\_Type](#) \*base)  
*Resets the ENET module.*

### MII interface operation

- void [ENET\\_SetMII](#) ([ENET\\_Type](#) \*base, [enet\\_mii\\_speed\\_t](#) speed, [enet\\_mii\\_duplex\\_t](#) duplex)  
*Sets the ENET MII speed and duplex.*
- void [ENET\\_SetSMI](#) ([ENET\\_Type](#) \*base, [uint32\\_t](#) srcClock\_Hz, bool isPreambleDisabled)  
*Sets the ENET SMI(serial management interface)- MII management interface.*
- static bool [ENET\\_GetSMI](#) ([ENET\\_Type](#) \*base)  
*Gets the ENET SMI- MII management interface configuration.*
- static [uint32\\_t](#) [ENET\\_ReadSMIData](#) ([ENET\\_Type](#) \*base)  
*Reads data from the PHY register through an SMI interface.*
- void [ENET\\_StartSMIRead](#) ([ENET\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) phyReg, [enet\\_mii\\_read\\_t](#) operation)  
*Starts an SMI (Serial Management Interface) read command.*
- void [ENET\\_StartSMIWrite](#) ([ENET\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) phyReg, [enet\\_mii\\_write\\_t](#) operation, [uint32\\_t](#) data)  
*Starts an SMI write command.*
- void [ENET\\_StartExtC45SMIRead](#) ([ENET\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) phyReg)  
*Starts the extended IEEE802.3 Clause 45 MDIO format SMI read command.*
- void [ENET\\_StartExtC45SMIWrite](#) ([ENET\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) phyReg, [uint32\\_t](#) data)  
*Starts the extended IEEE802.3 Clause 45 MDIO format SMI write command.*

### MAC Address Filter

- void [ENET\\_SetMacAddr](#) ([ENET\\_Type](#) \*base, [uint8\\_t](#) \*macAddr)  
*Sets the ENET module Mac address.*
- void [ENET\\_GetMacAddr](#) ([ENET\\_Type](#) \*base, [uint8\\_t](#) \*macAddr)  
*Gets the ENET module Mac address.*
- void [ENET\\_AddMulticastGroup](#) ([ENET\\_Type](#) \*base, [uint8\\_t](#) \*address)  
*Adds the ENET device to a multicast group.*
- void [ENET\\_LeaveMulticastGroup](#) ([ENET\\_Type](#) \*base, [uint8\\_t](#) \*address)  
*Moves the ENET device from a multicast group.*

### Other basic operation

- static void [ENET\\_ActiveRead](#) ([ENET\\_Type](#) \*base)  
*Activates ENET read or receive.*
- static void [ENET\\_EnableSleepMode](#) ([ENET\\_Type](#) \*base, bool enable)  
*Enables/disables the MAC to enter sleep mode.*



- static void [ENET\\_GetAccelFunction](#) (ENET\_Type \*base, uint32\_t \*txAccelOption, uint32\_t \*rxAccelOption)  
*Gets ENET transmit and receive accelerator functions from MAC controller.*

## Interrupts.

- static void [ENET\\_EnableInterrupts](#) (ENET\_Type \*base, uint32\_t mask)  
*Enables the ENET interrupt.*
- static void [ENET\\_DisableInterrupts](#) (ENET\_Type \*base, uint32\_t mask)  
*Disables the ENET interrupt.*
- static uint32\_t [ENET\\_GetInterruptStatus](#) (ENET\_Type \*base)  
*Gets the ENET interrupt status flag.*
- static void [ENET\\_ClearInterruptStatus](#) (ENET\_Type \*base, uint32\_t mask)  
*Clears the ENET interrupt events status flag.*

## Transactional operation

- void [ENET\\_SetCallback](#) (enet\_handle\_t \*handle, [enet\\_callback\\_t](#) callback, void \*userData)  
*Sets the callback function.*
- void [ENET\\_GetRxErrBeforeReadFrame](#) (enet\_handle\_t \*handle, [enet\\_data\\_error\\_stats\\_t](#) \*eError-Static)  
*Gets the error statistics of a received frame for ENET single ring.*
- status\_t [ENET\\_GetTxErrAfterSendFrame](#) (enet\_handle\_t \*handle, [enet\\_data\\_error\\_stats\\_t](#) \*eError-Static)  
*Gets the ENET transmit frame statistics after the data send for single ring.*
- status\_t [ENET\\_GetRxFrameSize](#) (enet\_handle\_t \*handle, uint32\_t \*length)  
*Gets the size of the read frame for single ring.*
- status\_t [ENET\\_ReadFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t \*data, uint32\_t length)  
*Reads a frame from the ENET device for single ring.*
- status\_t [ENET\\_SendFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, const uint8\_t \*data, uint32\_t length)  
*Transmits an ENET frame for single ring.*
- void [ENET\\_TransmitIRQHandler](#) (ENET\_Type \*base, enet\_handle\_t \*handle)  
*The transmit IRQ handler.*
- void [ENET\\_ReceiveIRQHandler](#) (ENET\_Type \*base, enet\_handle\_t \*handle)  
*The receive IRQ handler.*
- void [ENET\\_ErrorIRQHandler](#) (ENET\_Type \*base, enet\_handle\_t \*handle)  
*Some special IRQ handler including the error, mii, wakeup irq handler.*
- void [ENET\\_CommonFrame0IRQHandler](#) (ENET\_Type \*base)  
*the common IRQ handler for the tx/rx/error etc irq handler.*

## ENET PTP 1588 function operation

- void [ENET\\_Ptp1588Configure](#) (ENET\_Type \*base, enet\_handle\_t \*handle, [enet\\_ptp\\_config\\_t](#) \*ptpConfig)  
*Configures the ENET PTP IEEE 1588 feature with the basic configuration.*
- void [ENET\\_Ptp1588StartTimer](#) (ENET\_Type \*base, uint32\_t ptpClkSrc)  
*Starts the ENET PTP 1588 Timer.*
- static void [ENET\\_Ptp1588StopTimer](#) (ENET\_Type \*base)

## Data Structure Documentation

- *Stops the ENET PTP 1588 Timer.*  
• void [ENET\\_Ptp1588AdjustTimer](#) (ENET\_Type \*base, uint32\_t corrIncrease, uint32\_t corrPeriod)  
*Adjusts the ENET PTP 1588 timer.*
- static void [ENET\\_Ptp1588SetChannelMode](#) (ENET\_Type \*base, [enet\\_ptp\\_timer\\_channel\\_t](#) channel, [enet\\_ptp\\_timer\\_channel\\_mode\\_t](#) mode, bool intEnable)  
*Sets the ENET PTP 1588 timer channel mode.*
- static void [ENET\\_Ptp1588SetChannelOutputPulseWidth](#) (ENET\_Type \*base, [enet\\_ptp\\_timer\\_channel\\_t](#) channel, bool isOutputLow, uint8\_t pulseWidth, bool intEnable)  
*Sets ENET PTP 1588 timer channel mode pulse width.*
- static void [ENET\\_Ptp1588SetChannelCmpValue](#) (ENET\_Type \*base, [enet\\_ptp\\_timer\\_channel\\_t](#) channel, uint32\_t cmpValue)  
*Sets the ENET PTP 1588 timer channel comparison value.*
- static bool [ENET\\_Ptp1588GetChannelStatus](#) (ENET\_Type \*base, [enet\\_ptp\\_timer\\_channel\\_t](#) channel)  
*Gets the ENET PTP 1588 timer channel status.*
- static void [ENET\\_Ptp1588ClearChannelStatus](#) (ENET\_Type \*base, [enet\\_ptp\\_timer\\_channel\\_t](#) channel)  
*Clears the ENET PTP 1588 timer channel status.*
- void [ENET\\_Ptp1588GetTimer](#) (ENET\_Type \*base, enet\_handle\_t \*handle, [enet\\_ptp\\_time\\_t](#) \*ptpTime)  
*Gets the current ENET time from the PTP 1588 timer.*
- void [ENET\\_Ptp1588SetTimer](#) (ENET\_Type \*base, enet\_handle\_t \*handle, [enet\\_ptp\\_time\\_t](#) \*ptpTime)  
*Sets the ENET PTP 1588 timer to the assigned time.*
- void [ENET\\_Ptp1588TimerIRQHandler](#) (ENET\_Type \*base, enet\_handle\_t \*handle)  
*The IEEE 1588 PTP time stamp interrupt handler.*
- status\_t [ENET\\_GetRxFrameTime](#) (enet\_handle\_t \*handle, [enet\\_ptp\\_time\\_data\\_t](#) \*ptpTimeData)  
*Gets the time stamp of the received frame.*
- status\_t [ENET\\_GetTxFrameTime](#) (enet\_handle\_t \*handle, [enet\\_ptp\\_time\\_data\\_t](#) \*ptpTimeData)  
*Gets the time stamp of the transmit frame.*

## 19.3 Data Structure Documentation

### 19.3.1 struct enet\_rx\_bd\_struct\_t

#### Data Fields

- uint16\_t [length](#)  
*Buffer descriptor data length.*
- uint16\_t [control](#)  
*Buffer descriptor control and status.*
- uint8\_t \* [buffer](#)  
*Data buffer pointer.*
- uint16\_t [controlExtend0](#)  
*Extend buffer descriptor control0.*
- uint16\_t [controlExtend1](#)  
*Extend buffer descriptor control1.*
- uint16\_t [payloadChecksum](#)  
*Internal payload checksum.*

- uint8\_t [headerLength](#)  
*Header length.*
- uint8\_t [protocolTyte](#)  
*Protocol type.*
- uint16\_t [controlExtend2](#)  
*Extend buffer descriptor control2.*
- uint32\_t [timestamp](#)  
*Timestamp.*

### 19.3.1.0.0.17 Field Documentation

- 19.3.1.0.0.17.1 uint16\_t enet\_rx\_bd\_struct\_t::length
- 19.3.1.0.0.17.2 uint16\_t enet\_rx\_bd\_struct\_t::control
- 19.3.1.0.0.17.3 uint8\_t\* enet\_rx\_bd\_struct\_t::buffer
- 19.3.1.0.0.17.4 uint16\_t enet\_rx\_bd\_struct\_t::controlExtend0
- 19.3.1.0.0.17.5 uint16\_t enet\_rx\_bd\_struct\_t::controlExtend1
- 19.3.1.0.0.17.6 uint16\_t enet\_rx\_bd\_struct\_t::payloadChecksum
- 19.3.1.0.0.17.7 uint8\_t enet\_rx\_bd\_struct\_t::headerLength
- 19.3.1.0.0.17.8 uint8\_t enet\_rx\_bd\_struct\_t::protocolTyte
- 19.3.1.0.0.17.9 uint16\_t enet\_rx\_bd\_struct\_t::controlExtend2
- 19.3.1.0.0.17.10 uint32\_t enet\_rx\_bd\_struct\_t::timestamp

### 19.3.2 struct enet\_tx\_bd\_struct\_t

#### Data Fields

- uint16\_t [length](#)  
*Buffer descriptor data length.*
- uint16\_t [control](#)  
*Buffer descriptor control and status.*
- uint8\_t \* [buffer](#)  
*Data buffer pointer.*
- uint16\_t [controlExtend0](#)  
*Extend buffer descriptor control0.*
- uint16\_t [controlExtend1](#)  
*Extend buffer descriptor control1.*
- uint16\_t [controlExtend2](#)  
*Extend buffer descriptor control2.*
- uint32\_t [timestamp](#)  
*Timestamp.*

## Data Structure Documentation

### 19.3.2.0.0.18 Field Documentation

19.3.2.0.0.18.1 `uint16_t enet_tx_bd_struct_t::length`

19.3.2.0.0.18.2 `uint16_t enet_tx_bd_struct_t::control`

19.3.2.0.0.18.3 `uint8_t* enet_tx_bd_struct_t::buffer`

19.3.2.0.0.18.4 `uint16_t enet_tx_bd_struct_t::controlExtend0`

19.3.2.0.0.18.5 `uint16_t enet_tx_bd_struct_t::controlExtend1`

19.3.2.0.0.18.6 `uint16_t enet_tx_bd_struct_t::controlExtend2`

19.3.2.0.0.18.7 `uint32_t enet_tx_bd_struct_t::timestamp`

### 19.3.3 `struct enet_data_error_stats_t`

#### Data Fields

- `uint32_t statsRxLenGreaterErr`  
*Receive length greater than RCR[MAX\_FL].*
- `uint32_t statsRxAlignErr`  
*Receive non-octet alignment/.*
- `uint32_t statsRxFcsErr`  
*Receive CRC error.*
- `uint32_t statsRxOverRunErr`  
*Receive over run.*
- `uint32_t statsRxTruncateErr`  
*Receive truncate.*
- `uint32_t statsRxProtocolChecksumErr`  
*Receive protocol checksum error.*
- `uint32_t statsRxIpHeadChecksumErr`  
*Receive IP header checksum error.*
- `uint32_t statsRxMacErr`  
*Receive Mac error.*
- `uint32_t statsRxPhyErr`  
*Receive PHY error.*
- `uint32_t statsRxCollisionErr`  
*Receive collision.*
- `uint32_t statsTxErr`  
*The error happen when transmit the frame.*
- `uint32_t statsTxFrameErr`  
*The transmit frame is error.*
- `uint32_t statsTxOverflowErr`  
*Transmit overflow.*
- `uint32_t statsTxLateCollisionErr`  
*Transmit late collision.*
- `uint32_t statsTxExcessCollisionErr`  
*Transmit excess collision.*

- uint32\_t [statsTxUnderFlowErr](#)  
*Transmit under flow error.*
- uint32\_t [statsTxTsErr](#)  
*Transmit time stamp error.*

#### 19.3.3.0.0.19 Field Documentation

- 19.3.3.0.0.19.1 uint32\_t enet\_data\_error\_stats\_t::statsRxLenGreaterErr
- 19.3.3.0.0.19.2 uint32\_t enet\_data\_error\_stats\_t::statsRxFcsErr
- 19.3.3.0.0.19.3 uint32\_t enet\_data\_error\_stats\_t::statsRxOverRunErr
- 19.3.3.0.0.19.4 uint32\_t enet\_data\_error\_stats\_t::statsRxTruncateErr
- 19.3.3.0.0.19.5 uint32\_t enet\_data\_error\_stats\_t::statsRxProtocolChecksumErr
- 19.3.3.0.0.19.6 uint32\_t enet\_data\_error\_stats\_t::statsRxIplHeadChecksumErr
- 19.3.3.0.0.19.7 uint32\_t enet\_data\_error\_stats\_t::statsRxMacErr
- 19.3.3.0.0.19.8 uint32\_t enet\_data\_error\_stats\_t::statsRxPhyErr
- 19.3.3.0.0.19.9 uint32\_t enet\_data\_error\_stats\_t::statsRxCollisionErr
- 19.3.3.0.0.19.10 uint32\_t enet\_data\_error\_stats\_t::statsTxErr
- 19.3.3.0.0.19.11 uint32\_t enet\_data\_error\_stats\_t::statsTxFrameErr
- 19.3.3.0.0.19.12 uint32\_t enet\_data\_error\_stats\_t::statsTxOverflowErr
- 19.3.3.0.0.19.13 uint32\_t enet\_data\_error\_stats\_t::statsTxLateCollisionErr
- 19.3.3.0.0.19.14 uint32\_t enet\_data\_error\_stats\_t::statsTxExcessCollisionErr
- 19.3.3.0.0.19.15 uint32\_t enet\_data\_error\_stats\_t::statsTxUnderFlowErr
- 19.3.3.0.0.19.16 uint32\_t enet\_data\_error\_stats\_t::statsTxTsErr

#### 19.3.4 struct enet\_buffer\_config\_t

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

1. The aligned receive and transmit buffer size must be evenly divisible by ENET\_BUFF\_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.
2. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET\_BUFF\_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
3. The aligned transmit and receive data buffer start address must be evenly divisible by ENET\_BUFF-

## Data Structure Documentation

F\_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber \* rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber \* txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.

### Data Fields

- `uint16_t rxBdNumber`  
*Receive buffer descriptor number.*
- `uint16_t txBdNumber`  
*Transmit buffer descriptor number.*
- `uint32_t rxBuffSizeAlign`  
*Aligned receive data buffer size.*
- `uint32_t txBuffSizeAlign`  
*Aligned transmit data buffer size.*
- `volatile enet_rx_bd_struct_t * rxBdStartAddrAlign`  
*Aligned receive buffer descriptor start address: should be non-cacheable.*
- `volatile enet_tx_bd_struct_t * txBdStartAddrAlign`  
*Aligned transmit buffer descriptor start address: should be non-cacheable.*
- `uint8_t * rxBufferAlign`  
*Receive data buffer start address.*
- `uint8_t * txBufferAlign`  
*Transmit data buffer start address.*

#### 19.3.4.0.0.20 Field Documentation

19.3.4.0.0.20.1 `uint16_t enet_buffer_config_t::rxBdNumber`

19.3.4.0.0.20.2 `uint16_t enet_buffer_config_t::txBdNumber`

19.3.4.0.0.20.3 `uint32_t enet_buffer_config_t::rxBuffSizeAlign`

19.3.4.0.0.20.4 `uint32_t enet_buffer_config_t::txBuffSizeAlign`

19.3.4.0.0.20.5 `volatile enet_rx_bd_struct_t* enet_buffer_config_t::rxBdStartAddrAlign`

19.3.4.0.0.20.6 `volatile enet_tx_bd_struct_t* enet_buffer_config_t::txBdStartAddrAlign`

19.3.4.0.0.20.7 `uint8_t* enet_buffer_config_t::rxBufferAlign`

19.3.4.0.0.20.8 `uint8_t* enet_buffer_config_t::txBufferAlign`

#### 19.3.5 struct `enet_ptp_time_t`

### Data Fields

- `uint64_t second`

*Second.*

- uint32\_t [nanosecond](#)  
*Nanosecond.*

### 19.3.5.0.0.21 Field Documentation

19.3.5.0.0.21.1 uint64\_t enet\_ptp\_time\_t::second

19.3.5.0.0.21.2 uint32\_t enet\_ptp\_time\_t::nanosecond

## 19.3.6 struct enet\_ptp\_time\_data\_t

### Data Fields

- uint8\_t [version](#)  
*PTP version.*
- uint8\_t [sourcePortId](#) [kENET\_PtpSrcPortIdLen]  
*PTP source port ID.*
- uint16\_t [sequenceId](#)  
*PTP sequence ID.*
- uint8\_t [messageType](#)  
*PTP message type.*
- [enet\\_ptp\\_time\\_t](#) [timeStamp](#)  
*PTP timestamp.*

### 19.3.6.0.0.22 Field Documentation

19.3.6.0.0.22.1 uint8\_t enet\_ptp\_time\_data\_t::version

19.3.6.0.0.22.2 uint8\_t enet\_ptp\_time\_data\_t::sourcePortId[kENET\_PtpSrcPortIdLen]

19.3.6.0.0.22.3 uint16\_t enet\_ptp\_time\_data\_t::sequenceId

19.3.6.0.0.22.4 uint8\_t enet\_ptp\_time\_data\_t::messageType

19.3.6.0.0.22.5 enet\_ptp\_time\_t enet\_ptp\_time\_data\_t::timeStamp

## 19.3.7 struct enet\_ptp\_time\_data\_ring\_t

### Data Fields

- uint32\_t [front](#)  
*The first index of the ring.*
- uint32\_t [end](#)  
*The end index of the ring.*
- uint32\_t [size](#)  
*The size of the ring.*
- [enet\\_ptp\\_time\\_data\\_t](#) \* [ptpTsData](#)  
*PTP message data structure.*

## Data Structure Documentation

### 19.3.7.0.0.23 Field Documentation

19.3.7.0.0.23.1 `uint32_t enet_ptp_time_data_ring_t::front`

19.3.7.0.0.23.2 `uint32_t enet_ptp_time_data_ring_t::end`

19.3.7.0.0.23.3 `uint32_t enet_ptp_time_data_ring_t::size`

19.3.7.0.0.23.4 `enet_ptp_time_data_t* enet_ptp_time_data_ring_t::ptpTsData`

### 19.3.8 `struct enet_ptp_config_t`

#### Data Fields

- `uint8_t ptpTsRxBuffNum`  
*Receive 1588 timestamp buffer number.*
- `uint8_t ptpTsTxBuffNum`  
*Transmit 1588 timestamp buffer number.*
- `enet_ptp_time_data_t * rxPtpTsData`  
*The start address of 1588 receive timestamp buffers.*
- `enet_ptp_time_data_t * txPtpTsData`  
*The start address of 1588 transmit timestamp buffers.*
- `enet_ptp_timer_channel_t channel`  
*Used for ERRATA\_2579: the PTP 1588 timer channel for time interrupt.*
- `uint32_t ptp1588ClockSrc_Hz`  
*The clock source of the PTP 1588 timer.*

### 19.3.8.0.0.24 Field Documentation

19.3.8.0.0.24.1 `enet_ptp_timer_channel_t enet_ptp_config_t::channel`

19.3.8.0.0.24.2 `uint32_t enet_ptp_config_t::ptp1588ClockSrc_Hz`

### 19.3.9 `struct enet_intcoalesce_config_t`

#### Data Fields

- `uint8_t txCoalesceFrameCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit interrupt coalescing frame count threshold.*
- `uint16_t txCoalesceTimeCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit interrupt coalescing timer count threshold.*
- `uint8_t rxCoalesceFrameCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive interrupt coalescing frame count threshold.*
- `uint16_t rxCoalesceTimeCount` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive interrupt coalescing timer count threshold.*



### 19.3.9.0.0.25 Field Documentation

- 19.3.9.0.0.25.1 `uint8_t enet_intcoalesce_config_t::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- 19.3.9.0.0.25.2 `uint16_t enet_intcoalesce_config_t::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`
- 19.3.9.0.0.25.3 `uint8_t enet_intcoalesce_config_t::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- 19.3.9.0.0.25.4 `uint16_t enet_intcoalesce_config_t::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

### 19.3.10 struct `enet_config_t`

Note:

1. `macSpecialConfig` is used for a special control configuration, A logical OR of "`enet_special_control_flag_t`". For a special configuration for MAC, set this parameter to 0.
2. `txWatermark` is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of `txWatermark` is 0x2F - 4032 bytes written to TX FIFO .... `txWatermark` allows minimizing the transmit latency to set the `txWatermark` to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
3. `rxFifoFullThreshold` is similar to the `txWatermark` for cut-through operation in RX. It is in 64-bit words. The minimum is `ENET_FIFO_MIN_RX_FULL` and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size is smaller than the `txWatermark`, the frame is still transmitted. The rule is the same for `rxFifoFullThreshold` in the receive direction.
4. When "`kENET_ControlFlowControlEnable`" is set in the `macSpecialConfig`, ensure that the `pauseDuration`, `rxFifoEmptyThreshold`, and `rxFifoStatEmptyThreshold` are set for flow control enabled case.
5. When "`kENET_ControlStoreAndFwdDisabled`" is set in the `macSpecialConfig`, ensure that the `rxFifoFullThreshold` and `txFifoWatermark` are set for store and forward disable.
6. The `rxAccelerConfig` and `txAccelerConfig` default setting with 0 - accelerator are disabled. The "`enet_tx_accelerator_t`" and "`enet_rx_accelerator_t`" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, `kENET_ControlStoreAndFwdDisabled` should not be set.
7. The `intCoalesceCf` can be used in the rx or tx enabled cases to decrease the CPU loading.

### Data Fields

- `uint32_t macSpecialConfig`  
*Mac special configuration.*
- `uint32_t interrupt`  
*Mac interrupt source.*

## Data Structure Documentation

- `uint16_t rxMaxFrameLen`  
*Receive maximum frame length.*
- `enet_mii_mode_t miiMode`  
*MII mode.*
- `enet_mii_speed_t miiSpeed`  
*MII Speed.*
- `enet_mii_duplex_t miiDuplex`  
*MII duplex.*
- `uint8_t rxAccelerConfig`  
*Receive accelerator, A logical OR of "enet\_rx\_accelerator\_t".*
- `uint8_t txAccelerConfig`  
*Transmit accelerator, A logical OR of "enet\_tx\_accelerator\_t".*
- `uint16_t pauseDuration`  
*For flow control enabled case: Pause duration.*
- `uint8_t rxFifoEmptyThreshold`  
*For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.*
- `uint8_t rxFifoFullThreshold`  
*For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.*
- `uint8_t txFifoWatermark`  
*For store and forward disable case, the data required in TX FIFO before a frame transmit start.*
- `uint8_t ringNum`  
*Number of used rings.*

### 19.3.10.0.0.26 Field Documentation

#### 19.3.10.0.0.26.1 `uint32_t enet_config_t::macSpecialConfig`

A logical OR of "enet\_special\_control\_flag\_t".

#### 19.3.10.0.0.26.2 `uint32_t enet_config_t::interrupt`

A logical OR of "enet\_interrupt\_enable\_t".

- 19.3.10.0.0.26.3 `uint16_t enet_config_t::rxMaxFrameLen`
- 19.3.10.0.0.26.4 `enet_mii_mode_t enet_config_t::miiMode`
- 19.3.10.0.0.26.5 `enet_mii_speed_t enet_config_t::miiSpeed`
- 19.3.10.0.0.26.6 `enet_mii_duplex_t enet_config_t::miiDuplex`
- 19.3.10.0.0.26.7 `uint8_t enet_config_t::rxAccelerConfig`
- 19.3.10.0.0.26.8 `uint8_t enet_config_t::txAccelerConfig`
- 19.3.10.0.0.26.9 `uint16_t enet_config_t::pauseDuration`
- 19.3.10.0.0.26.10 `uint8_t enet_config_t::rxFifoEmptyThreshold`
- 19.3.10.0.0.26.11 `uint8_t enet_config_t::rxFifoFullThreshold`
- 19.3.10.0.0.26.12 `uint8_t enet_config_t::txFifoWatermark`
- 19.3.10.0.0.26.13 `uint8_t enet_config_t::ringNum`

default with 1 – single ring.

## 19.3.11 `struct enet_handle`

### Data Fields

- volatile `enet_rx_bd_struct_t * rxBdBase` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer descriptor base address pointer.*
- volatile `enet_rx_bd_struct_t * rxBdCurrent` [FSL\_FEATURE\_ENET\_QUEUE]  
*The current available receive buffer descriptor pointer.*
- volatile `enet_tx_bd_struct_t * txBdBase` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer descriptor base address pointer.*
- volatile `enet_tx_bd_struct_t * txBdCurrent` [FSL\_FEATURE\_ENET\_QUEUE]  
*The current available transmit buffer descriptor pointer.*
- `uint32_t rxBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer size alignment.*
- `uint32_t txBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer size alignment.*
- `uint8_t ringNum`  
*Number of used rings.*
- `enet_callback_t callback`  
*Callback function.*
- `void * userData`  
*Callback function parameter.*
- volatile `enet_tx_bd_struct_t * txBdDirtyStatic` [FSL\_FEATURE\_ENET\_QUEUE]  
*The dirty transmit buffer descriptor for error static update.*
- volatile `enet_tx_bd_struct_t * txBdDirtyTime` [FSL\_FEATURE\_ENET\_QUEUE]

## Macro Definition Documentation

- The dirty transmit buffer descriptor for time stamp update.*
- `uint64_t msTimerSecond`  
*The second for Master PTP timer .*
- `enet_ptp_time_data_ring_t rxPtpTsDataRing`  
*Receive PTP 1588 time stamp data ring buffer.*
- `enet_ptp_time_data_ring_t txPtpTsDataRing`  
*Transmit PTP 1588 time stamp data ring buffer.*

### 19.3.11.0.0.27 Field Documentation

- 19.3.11.0.0.27.1 `volatile enet_rx_bd_struct_t* enet_handle_t::rxBdBase[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.2 `volatile enet_rx_bd_struct_t* enet_handle_t::rxBdCurrent[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.3 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdBase[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.4 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdCurrent[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.5 `uint32_t enet_handle_t::rxBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.6 `uint32_t enet_handle_t::txBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.7 `uint8_t enet_handle_t::ringNum`
- 19.3.11.0.0.27.8 `enet_callback_t enet_handle_t::callback`
- 19.3.11.0.0.27.9 `void* enet_handle_t::userData`
- 19.3.11.0.0.27.10 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdDirtyStatic[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.11 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdDirtyTime[FSL_FEATURE_ENET_QUEUE]`
- 19.3.11.0.0.27.12 `uint64_t enet_handle_t::msTimerSecond`
- 19.3.11.0.0.27.13 `enet_ptp_time_data_ring_t enet_handle_t::rxPtpTsDataRing`
- 19.3.11.0.0.27.14 `enet_ptp_time_data_ring_t enet_handle_t::txPtpTsDataRing`

## 19.4 Macro Definition Documentation

- 19.4.1 `#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))`

Version 2.2.3.



## Macro Definition Documentation

- 19.4.2 **#define ENET\_BUFFDESCRIPTOR\_RX\_EMPTY\_MASK 0x8000U**
- 19.4.3 **#define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER1\_MASK 0x4000U**
- 19.4.4 **#define ENET\_BUFFDESCRIPTOR\_RX\_WRAP\_MASK 0x2000U**
- 19.4.5 **#define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER2\_Mask 0x1000U**
- 19.4.6 **#define ENET\_BUFFDESCRIPTOR\_RX\_LAST\_MASK 0x0800U**
- 19.4.7 **#define ENET\_BUFFDESCRIPTOR\_RX\_MISS\_MASK 0x0100U**
- 19.4.8 **#define ENET\_BUFFDESCRIPTOR\_RX\_BROADCAST\_MASK 0x0080U**
- 19.4.9 **#define ENET\_BUFFDESCRIPTOR\_RX\_MULTICAST\_MASK 0x0040U**
- 19.4.10 **#define ENET\_BUFFDESCRIPTOR\_RX\_LENVIOLATE\_MASK 0x0020U**
- 19.4.11 **#define ENET\_BUFFDESCRIPTOR\_RX\_NOOCTET\_MASK 0x0010U**
- 19.4.12 **#define ENET\_BUFFDESCRIPTOR\_RX\_CRC\_MASK 0x0004U**
- 19.4.13 **#define ENET\_BUFFDESCRIPTOR\_RX\_OVERRUN\_MASK 0x0002U**
- 19.4.14 **#define ENET\_BUFFDESCRIPTOR\_RX\_TRUNC\_MASK 0x0001U**
- 19.4.15 **#define ENET\_BUFFDESCRIPTOR\_TX\_READY\_MASK 0x8000U**
- 19.4.16 **#define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWENER1\_MASK 0x4000U**
- 19.4.17 **#define ENET\_BUFFDESCRIPTOR\_TX\_WRAP\_MASK 0x2000U**
- 19.4.18 **#define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWENER2\_MASK 0x1000U**
- 19.4.19 **#define ENET\_BUFFDESCRIPTOR\_TX\_LAST\_MASK 0x0800U**
- 19.4.20 **#define ENET\_BUFFDESCRIPTOR\_TX\_TRANMITCRC\_MASK 0x0400U**
- 19.4.21 **#define ENET\_BUFFDESCRIPTOR\_RX\_IPV4\_MASK 0x0001U**

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

**19.4.41 #define ENET\_FRAME\_MAX\_FRAMELEN 1518U**

**19.4.42 #define ENET\_FIFO\_MIN\_RX\_FULL 5U**

**19.4.43 #define ENET\_RX\_MIN\_BUFFERSIZE 256U**

## 19.5 Typedef Documentation

**19.5.1 typedef void(\* enet\_callback\_t)(ENET\_Type \*base, enet\_handle\_t \*handle, enet\_event\_t event, void \*userData)**

## 19.6 Enumeration Type Documentation

### 19.6.1 enum \_enet\_status

Enumerator

*kStatus\_ENET\_RxFrameError* A frame received but data error happen.  
*kStatus\_ENET\_RxFrameFail* Failed to receive a frame.  
*kStatus\_ENET\_RxFrameEmpty* No frame arrive.  
*kStatus\_ENET\_TxFrameOverLen* Tx frame over length.  
*kStatus\_ENET\_TxFrameBusy* Tx buffer descriptors are under process.  
*kStatus\_ENET\_TxFrameFail* Transmit frame fail.  
*kStatus\_ENET\_PtpTsRingFull* Timestamp ring full.  
*kStatus\_ENET\_PtpTsRingEmpty* Timestamp ring empty.

### 19.6.2 enum enet\_mii\_mode\_t

Enumerator

*kENET\_MiiMode* MII mode for data interface.  
*kENET\_RmiiMode* RMII mode for data interface.

### 19.6.3 enum enet\_mii\_speed\_t

Notice: "kENET\_MiiSpeed1000M" only supported when mii mode is "kENET\_RgmiiMode".

## Enumeration Type Documentation

Enumerator

*kENET\_MiiSpeed10M* Speed 10 Mbps.  
*kENET\_MiiSpeed100M* Speed 100 Mbps.

### 19.6.4 enum enet\_mii\_duplex\_t

Enumerator

*kENET\_MiiHalfDuplex* Half duplex mode.  
*kENET\_MiiFullDuplex* Full duplex mode.

### 19.6.5 enum enet\_mii\_write\_t

Enumerator

*kENET\_MiiWriteNoCompliant* Write frame operation, but not MII-compliant.  
*kENET\_MiiWriteValidFrame* Write frame operation for a valid MII management frame.

### 19.6.6 enum enet\_mii\_read\_t

Enumerator

*kENET\_MiiReadValidFrame* Read frame operation for a valid MII management frame.  
*kENET\_MiiReadNoCompliant* Read frame operation, but not MII-compliant.

### 19.6.7 enum enet\_mii\_extend\_opcode

Enumerator

*kENET\_MiiAddrWrite\_C45* Address Write operation.  
*kENET\_MiiWriteFrame\_C45* Write frame operation for a valid MII management frame.  
*kENET\_MiiReadFrame\_C45* Read frame operation for a valid MII management frame.

### 19.6.8 enum enet\_special\_control\_flag\_t

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



Enumerator

*kENET\_ControlFlowControlEnable* Enable ENET flow control: pause frame.  
*kENET\_ControlRxPayloadCheckEnable* Enable ENET receive payload length check.  
*kENET\_ControlRxPadRemoveEnable* Padding is removed from received frames.  
*kENET\_ControlRxBroadCastRejectEnable* Enable broadcast frame reject.  
*kENET\_ControlMacAddrInsert* Enable MAC address insert.  
*kENET\_ControlStoreAndFwdDisable* Enable FIFO store and forward.  
*kENET\_ControlSMIPreambleDisable* Enable SMI preamble.  
*kENET\_ControlPromiscuousEnable* Enable promiscuous mode.  
*kENET\_ControlMIILoopEnable* Enable ENET MII loop back.  
*kENET\_ControlVLANTagEnable* Enable normal VLAN (single vlan tag).

### 19.6.9 enum enet\_interrupt\_enable\_t

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

Enumerator

*kENET\_BabrInterrupt* Babbling receive error interrupt source.  
*kENET\_BabtInterrupt* Babbling transmit error interrupt source.  
*kENET\_GraceStopInterrupt* Graceful stop complete interrupt source.  
*kENET\_TxFrameInterrupt* TX FRAME interrupt source.  
*kENET\_TxBufferInterrupt* TX BUFFER interrupt source.  
*kENET\_RxFrameInterrupt* RX FRAME interrupt source.  
*kENET\_RxBufferInterrupt* RX BUFFER interrupt source.  
*kENET\_MiiInterrupt* MII interrupt source.  
*kENET\_EBusERInterrupt* Ethernet bus error interrupt source.  
*kENET\_LateCollisionInterrupt* Late collision interrupt source.  
*kENET\_RetryLimitInterrupt* Collision Retry Limit interrupt source.  
*kENET\_UnderrunInterrupt* Transmit FIFO underrun interrupt source.  
*kENET\_PayloadRxInterrupt* Payload Receive error interrupt source.  
*kENET\_WakeupInterrupt* WAKEUP interrupt source.  
*kENET\_TsAvailInterrupt* TS AVAIL interrupt source for PTP.  
*kENET\_TsTimerInterrupt* TS WRAP interrupt source for PTP.

### 19.6.10 enum enet\_event\_t

Enumerator

*kENET\_RxEvent* Receive event.  
*kENET\_TxEvent* Transmit event.  
*kENET\_ErrEvent* Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

## Enumeration Type Documentation

*kENET\_WakeUpEvent* Wake up from sleep mode event.  
*kENET\_TimeStampEvent* Time stamp event.  
*kENET\_TimeStampAvailEvent* Time stamp available event.

### 19.6.11 enum enet\_tx\_accelerator\_t

Enumerator

*kENET\_TxAccelIsShift16Enabled* Transmit FIFO shift-16.  
*kENET\_TxAccelIpCheckEnabled* Insert IP header checksum.  
*kENET\_TxAccelProtoCheckEnabled* Insert protocol checksum.

### 19.6.12 enum enet\_rx\_accelerator\_t

Enumerator

*kENET\_RxAccelPadRemoveEnabled* Padding removal for short IP frames.  
*kENET\_RxAccelIpCheckEnabled* Discard with wrong IP header checksum.  
*kENET\_RxAccelProtoCheckEnabled* Discard with wrong protocol checksum.  
*kENET\_RxAccelMacCheckEnabled* Discard with Mac layer errors.  
*kENET\_RxAccelIsShift16Enabled* Receive FIFO shift-16.

### 19.6.13 enum enet\_ptp\_event\_type\_t

Enumerator

*kENET\_PtpEventMsgType* PTP event message type.  
*kENET\_PtpSrcPortIdLen* PTP message sequence id length.  
*kENET\_PtpEventPort* PTP event port number.  
*kENET\_PtpGnrlPort* PTP general port number.

### 19.6.14 enum enet\_ptp\_timer\_channel\_t

Enumerator

*kENET\_PtpTimerChannel1* IEEE 1588 PTP timer Channel 1.  
*kENET\_PtpTimerChannel2* IEEE 1588 PTP timer Channel 2.  
*kENET\_PtpTimerChannel3* IEEE 1588 PTP timer Channel 3.  
*kENET\_PtpTimerChannel4* IEEE 1588 PTP timer Channel 4.

## 19.6.15 enum enet\_ptp\_timer\_channel\_mode\_t

Enumerator

- kENET\_PtpChannelDisable* Disable timer channel.
- kENET\_PtpChannelRisingCapture* Input capture on rising edge.
- kENET\_PtpChannelFallingCapture* Input capture on falling edge.
- kENET\_PtpChannelBothCapture* Input capture on both edges.
- kENET\_PtpChannelSoftCompare* Output compare software only.
- kENET\_PtpChannelToggleCompare* Toggle output on compare.
- kENET\_PtpChannelClearCompare* Clear output on compare.
- kENET\_PtpChannelSetCompare* Set output on compare.
- kENET\_PtpChannelClearCompareSetOverflow* Clear output on compare, set output on overflow.
- kENET\_PtpChannelSetCompareClearOverflow* Set output on compare, clear output on overflow.
- kENET\_PtpChannelPulseLowonCompare* Pulse output low on compare for one IEEE 1588 clock cycle.
- kENET\_PtpChannelPulseHighonCompare* Pulse output high on compare for one IEEE 1588 clock cycle.

## 19.7 Function Documentation

### 19.7.1 void ENET\_GetDefaultConfig ( enet\_config\_t \* config )

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

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

Parameters

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

### 19.7.2 void ENET\_Init ( ENET\_Type \* base, enet\_handle\_t \* handle, const enet\_config\_t \* config, const enet\_buffer\_config\_t \* bufferConfig, uint8\_t \* macAddr, uint32\_t srcClock\_Hz )

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

## Function Documentation

### Parameters

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

### Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" and calling [ENET\\_Ptp1588Configure\(\)](#) to configure the 1588 feature and related buffers after calling [ENET\\_Init\(\)](#).

### 19.7.3 void ENET\_Deinit ( ENET\_Type \* *base* )

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

### Parameters

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

### 19.7.4 static void ENET\_Reset ( ENET\_Type \* *base* ) [inline], [static]

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

Parameters

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

### 19.7.5 void ENET\_SetMII ( ENET\_Type \* *base*, enet\_mii\_speed\_t *speed*, enet\_mii\_duplex\_t *duplex* )

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

Parameters

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

### 19.7.6 void ENET\_SetSMI ( ENET\_Type \* *base*, uint32\_t *srcClock\_Hz*, bool *isPreambleDisabled* )

Parameters

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

### 19.7.7 static bool ENET\_GetSMI ( ENET\_Type \* *base* ) [inline], [static]

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

Parameters

---

## Function Documentation

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

Returns

The SMI setup status true or false.

**19.7.8 static uint32\_t ENET\_ReadSMIData ( ENET\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The data read from PHY

**19.7.9 void ENET\_StartSMIRead ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_read\_t *operation* )**

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

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

**19.7.10 void ENET\_StartSMIWrite ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_write\_t *operation*, uint32\_t *data* )**

Used for standard IEEE802.3 MDIO Clause 22 format.

## Parameters

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

### 19.7.11 void ENET\_StartExtC45SMIRead ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg* )

## Parameters

|                |                                                                                                                                                                                                           |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                                                                                                                             |
| <i>phyAddr</i> | The PHY address.                                                                                                                                                                                          |
| <i>phyReg</i>  | The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register):<br>phyReg = (devaddr << 16)   regAddr. |

### 19.7.12 void ENET\_StartExtC45SMIWrite ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, uint32\_t *data* )

## Parameters

|                |                                                                                                                                                                                                           |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                                                                                                                             |
| <i>phyAddr</i> | The PHY address.                                                                                                                                                                                          |
| <i>phyReg</i>  | The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register):<br>phyReg = (devaddr << 16)   regAddr. |
| <i>data</i>    | The data written to PHY.                                                                                                                                                                                  |

### 19.7.13 void ENET\_SetMacAddr ( ENET\_Type \* *base*, uint8\_t \* *macAddr* )

## Function Documentation

Parameters

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

### 19.7.14 void ENET\_GetMacAddr ( ENET\_Type \* *base*, uint8\_t \* *macAddr* )

Parameters

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

### 19.7.15 void ENET\_AddMulticastGroup ( ENET\_Type \* *base*, uint8\_t \* *address* )

Parameters

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

### 19.7.16 void ENET\_LeaveMulticastGroup ( ENET\_Type \* *base*, uint8\_t \* *address* )

Parameters

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

### 19.7.17 static void ENET\_ActiveRead ( ENET\_Type \* *base* ) [inline], [static]

This function is to active the enet read process. It is used for single descriptor ring/queue.



## Parameters

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

## Note

This must be called after the MAC configuration and state are ready. It must be called after the [ENET\\_Init\(\)](#) and [ENET\\_Ptp1588Configure\(\)](#). This should be called when the ENET receive required.

### 19.7.18 static void ENET\_EnableSleepMode ( ENET\_Type \* *base*, bool *enable* ) [inline], [static]

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

## Parameters

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

### 19.7.19 static void ENET\_GetAccelFunction ( ENET\_Type \* *base*, uint32\_t \* *txAccelOption*, uint32\_t \* *rxAccelOption* ) [inline], [static]

## Parameters

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

### 19.7.20 static void ENET\_EnableInterrupts ( ENET\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

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

## Function Documentation

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

## Parameters

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

### 19.7.21 `static void ENET_DisableInterrupts ( ENET_Type * base, uint32_t mask )` **[inline], [static]**

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

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

## Parameters

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

### 19.7.22 `static uint32_t ENET_GetInterruptStatus ( ENET_Type * base )` **[inline], [static]**

## Parameters

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

## Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: `enet_interrupt_enable_t`.

### 19.7.23 `static void ENET_ClearInterruptStatus ( ENET_Type * base, uint32_t mask )` **[inline], [static]**

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

## Function Documentation

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

### Parameters

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

### 19.7.24 void ENET\_SetCallback ( enet\_handle\_t \* handle, enet\_callback\_t callback, void \* userData )

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

### Parameters

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

### 19.7.25 void ENET\_GetRxErrBeforeReadFrame ( enet\_handle\_t \* handle, enet\_data\_error\_stats\_t \* eErrorStatic )

This API must be called after the `ENET_GetRxFrameSize` and before the `ENET_ReadFrame()`. If the `ENET_GetRxFrameSize` returns `kStatus_ENET_RxFrameError`, the `ENET_GetRxErrBeforeReadFrame` can be used to get the exact error statistics. This is an example.

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

Parameters

|                     |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <i>handle</i>       | The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init. |
| <i>eErrorStatic</i> | The error statistics structure pointer.                                                     |

**19.7.26 status\_t ENET\_GetTxErrAfterSendFrame ( enet\_handle\_t \* *handle*, enet\_data\_error\_stats\_t \* *eErrorStatic* )**

This interface gets the error statistics of the transmit frame. Because the error information is reported by the uDMA after the data delivery, this interface should be called after the data transmit API. It is recommended to call this function on transmit interrupt handler. After calling the ENET\_SendFrame, the transmit interrupt notifies the transmit completion.

Parameters

|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <i>handle</i>       | The PTP handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>eErrorStatic</i> | The error statistics structure pointer.                                          |

Returns

The execute status.

**19.7.27 status\_t ENET\_GetRxFrameSize ( enet\_handle\_t \* *handle*, uint32\_t \* *length* )**

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

Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET\_GetRxFrameSize, [ENET\\_ReadFrame\(\)](#) should be called to update the receive buffers If the result is not "kStatus\_ENET\_RxFrameEmpty".

Parameters

## Function Documentation

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>handle</i> | The ENET handler structure. This is the same handler pointer used in the ENET_Init. |
| <i>length</i> | The length of the valid frame received.                                             |

### Return values

|                                   |                                                                                                                                      |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_ENET_RxFrame-Empty</i> | No frame received. Should not call ENET_ReadFrame to read frame.                                                                     |
| <i>kStatus_ENET_RxFrame-Error</i> | Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.                    |
| <i>kStatus_Success</i>            | Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input. |

### 19.7.28 `status_t ENET_ReadFrame ( ENET_Type * base, enet_handle_t * handle, uint8_t * data, uint32_t length )`

This function reads a frame (both the data and the length) from the ENET buffer descriptors. The ENET\_GetRxFrameSize should be used to get the size of the prepared data buffer. This is an example:

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

### Parameters

---

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

Returns

The execute status, successful or failure.

**19.7.29** `status_t ENET_SendFrame ( ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length )`

Note

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

Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                     |
| <i>handle</i> | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>data</i>   | The data buffer provided by user to be send.                                      |
| <i>length</i> | The length of the data to be send.                                                |

Return values

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

**19.7.30** `void ENET_TransmitIRQHandler ( ENET_Type * base, enet_handle_t * handle )`

## Function Documentation

Parameters

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

### 19.7.31 void ENET\_ReceiveIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

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

### 19.7.32 void ENET\_ErrorIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

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

### 19.7.33 void ENET\_CommonFrame0IRQHandler ( ENET\_Type \* *base* )

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

Parameters

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

### 19.7.34 void ENET\_Ptp1588Configure ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_ptp\_config\_t \* *ptpConfig* )

The function sets the clock for PTP 1588 timer and enables time stamp interrupts and transmit interrupts for PTP 1588 features. This API should be called when the 1588 feature is enabled or the ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE is defined. ENET\_Init should be called before calling this API.



## Note

The PTP 1588 time-stamp second increase through time-stamp interrupt handler and the transmit time-stamp store is done through transmit interrupt handler. As a result, the TS interrupt and TX interrupt are enabled when you call this API.

## Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | ENET peripheral base address.   |
| <i>handle</i>    | ENET handler pointer.           |
| <i>ptpConfig</i> | The ENET PTP1588 configuration. |

### 19.7.35 void ENET\_Ptp1588StartTimer ( ENET\_Type \* *base*, uint32\_t *ptpClkSrc* )

This function is used to initialize the PTP timer. After the PTP starts, the PTP timer starts running.

## Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>base</i>      | ENET peripheral base address.      |
| <i>ptpClkSrc</i> | The clock source of the PTP timer. |

### 19.7.36 static void ENET\_Ptp1588StopTimer ( ENET\_Type \* *base* ) [inline], [static]

This function is used to stops the ENET PTP timer.

## Parameters

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

### 19.7.37 void ENET\_Ptp1588AdjustTimer ( ENET\_Type \* *base*, uint32\_t *corrIncrease*, uint32\_t *corrPeriod* )

## Parameters

---

## Function Documentation

|                     |                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ENET peripheral base address.                                                                                                                                                                                                                              |
| <i>corrIncrease</i> | The correction increment value. This value is added every time the correction timer expires. A value less than the PTP timer frequency(1/ptpClkSrc) slows down the timer, a value greater than the 1/ptpClkSrc speeds up the timer.                        |
| <i>corrPeriod</i>   | The PTP timer correction counter wrap-around value. This defines after how many timer clock the correction counter should be reset and trigger a correction increment on the timer. A value of 0 disables the correction counter and no correction occurs. |

**19.7.38 static void ENET\_Ptp1588SetChannelMode ( ENET\_Type \* *base*, enet\_ptp\_timer\_channel\_t *channel*, enet\_ptp\_timer\_channel\_mode\_t *mode*, bool *intEnable* ) [inline], [static]**

Parameters

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                                    |
| <i>channel</i>   | The ENET PTP timer channel number.                               |
| <i>mode</i>      | The PTP timer channel mode, see "enet_ptp_timer_channel_mode_t". |
| <i>intEnable</i> | Enables or disables the interrupt.                               |

**19.7.39 static void ENET\_Ptp1588SetChannelOutputPulseWidth ( ENET\_Type \* *base*, enet\_ptp\_timer\_channel\_t *channel*, bool *isOutputLow*, uint8\_t *pulseWidth*, bool *intEnable* ) [inline], [static]**

For the input "mode" in ENET\_Ptp1588SetChannelMode, the kENET\_PtpChannelPulseLowonCompare kENET\_PtpChannelPulseHighonCompare only support the pulse width for one 1588 clock. this function is extended for control the pulse width from 1 to 32 1588 clock cycles. so call this function if you need to set the timer channel mode for kENET\_PtpChannelPulseLowonCompare or kENET\_PtpChannelPulse-HighonCompare with pulse width more than one 1588 clock,

Parameters

|                    |                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | ENET peripheral base address.                                                                                                                     |
| <i>channel</i>     | The ENET PTP timer channel number.                                                                                                                |
| <i>isOutputLow</i> | True — timer channel is configured for output compare pulse output low. false — timer channel is configured for output compare pulse output high. |
| <i>pulseWidth</i>  | The pulse width control value, range from 0 ~ 31. 0 — pulse width is one 1588 clock cycle. 31 — pulse width is thirty two 1588 clock cycles.      |
| <i>intEnable</i>   | Enables or disables the interrupt.                                                                                                                |

**19.7.40** `static void ENET_Ptp1588SetChannelCmpValue ( ENET_Type * base,  
enet_ptp_timer_channel_t channel, uint32_t cmpValue ) [inline],  
[static]`

Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>base</i>     | ENET peripheral base address.                          |
| <i>channel</i>  | The PTP timer channel, see "enet_ptp_timer_channel_t". |
| <i>cmpValue</i> | The compare value for the compare setting.             |

**19.7.41** `static bool ENET_Ptp1588GetChannelStatus ( ENET_Type * base,  
enet_ptp_timer_channel_t channel ) [inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | ENET peripheral base address.       |
| <i>channel</i> | The IEEE 1588 timer channel number. |

Returns

True or false, Compare or capture operation status

**19.7.42** `static void ENET_Ptp1588ClearChannelStatus ( ENET_Type * base,  
enet_ptp_timer_channel_t channel ) [inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | ENET peripheral base address.       |
| <i>channel</i> | The IEEE 1588 timer channel number. |

**19.7.43** `void ENET_Ptp1588GetTimer ( ENET_Type * base, enet_handle_t * handle,  
enet_ptp_time_t * ptpTime )`

## Function Documentation

### Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                 |
| <i>handle</i>  | The ENET state pointer. This is the same state pointer used in the ENET_Init. |
| <i>ptpTime</i> | The PTP timer structure.                                                      |

**19.7.44 void ENET\_Ptp1588SetTimer ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_ptp\_time\_t \* *ptpTime* )**

### Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                 |
| <i>handle</i>  | The ENET state pointer. This is the same state pointer used in the ENET_Init. |
| <i>ptpTime</i> | The timer to be set to the PTP timer.                                         |

**19.7.45 void ENET\_Ptp1588TimerIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )**

### Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                 |
| <i>handle</i> | The ENET state pointer. This is the same state pointer used in the ENET_Init. |

**19.7.46 status\_t ENET\_GetRxFrameTime ( enet\_handle\_t \* *handle*, enet\_ptp\_time\_data\_t \* *ptpTimeData* )**

This function is used for PTP stack to get the timestamp captured by the ENET driver.

### Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>handle</i>      | The ENET handler pointer. This is the same state pointer used in ENET_Init. |
| <i>ptpTimeData</i> | The special PTP timestamp data for search the receive timestamp.            |

Return values

|                                     |                             |
|-------------------------------------|-----------------------------|
| <i>kStatus_Success</i>              | Get 1588 timestamp success. |
| <i>kStatus_ENET_PtpTs-RingEmpty</i> | 1588 timestamp ring empty.  |
| <i>kStatus_ENET_PtpTs-RingFull</i>  | 1588 timestamp ring full.   |

### 19.7.47 status\_t ENET\_GetTxFrameTime ( enet\_handle\_t \* handle, enet\_ptp\_time\_data\_t \* ptpTimeData )

This function is used for PTP stack to get the timestamp captured by the ENET driver.

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>handle</i>      | The ENET handler pointer. This is the same state pointer used in ENET_Init. |
| <i>ptpTimeData</i> | The special PTP timestamp data for search the receive timestamp.            |

Return values

|                                     |                             |
|-------------------------------------|-----------------------------|
| <i>kStatus_Success</i>              | Get 1588 timestamp success. |
| <i>kStatus_ENET_PtpTs-RingEmpty</i> | 1588 timestamp ring empty.  |
| <i>kStatus_ENET_PtpTs-RingFull</i>  | 1588 timestamp ring full.   |



## Chapter 20

# EWM: External Watchdog Monitor Driver

### 20.1 Overview

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

### 20.2 Typical use case

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

### Data Structures

- struct `ewm_config_t`  
*Describes EWM clock source. [More...](#)*

### Enumerations

- enum `_ewm_interrupt_enable_t` { `kEWM_InterruptEnable` = `EWM_CTRL_INTEN_MASK` }  
*EWM interrupt configuration structure with default settings all disabled.*
- enum `_ewm_status_flags_t` { `kEWM_RunningFlag` = `EWM_CTRL_EWMEN_MASK` }  
*EWM status flags.*

### Driver version

- `#define FSL_EWM_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 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`)  
*Disables the EWM interrupt.*
- static `uint32_t EWM_GetStatusFlags` (`EWM_Type *base`)  
*Gets all status flags.*
- void `EWM_Refresh` (`EWM_Type *base`)  
*Services the EWM.*

## Enumeration Type Documentation

### 20.3 Data Structure Documentation

#### 20.3.1 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.*

### 20.4 Macro Definition Documentation

#### 20.4.1 #define FSL\_EWM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 20.5 Enumeration Type Documentation

#### 20.5.1 enum \_ewm\_interrupt\_enable\_t

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

Enumerator

***kEWM\_InterruptEnable*** Enable the EWM to generate an interrupt.

#### 20.5.2 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.



## 20.6 Function Documentation

### 20.6.1 void EWM\_Init ( EWM\_Type \* *base*, const ewm\_config\_t \* *config* )

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
* ewm_config_t config;
* EWM_GetDefaultConfig(&config);
* config.compareHighValue = 0xAAU;
* EWM_Init(ewm_base,&config);
*
```

#### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | EWM peripheral base address  |
| <i>config</i> | The configuration of the EWM |

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

This function is used to shut down the EWM.

#### Parameters

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

### 20.6.3 void EWM\_GetDefaultConfig ( ewm\_config\_t \* *config* )

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
* ewmConfig->enableEwm = true;
* ewmConfig->enableEwmInput = false;
* ewmConfig->setInputAssertLogic = false;
* ewmConfig->enableInterrupt = false;
* ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
* ewmConfig->prescaler = 0;
* ewmConfig->compareLowValue = 0;
* ewmConfig->compareHighValue = 0xFEU;
*
```

## Function Documentation

Parameters

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

See Also

[ewm\\_config\\_t](#)

### 20.6.4 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> |

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

This function enables the EWM interrupt.

Parameters

|             |                                                                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | EWM peripheral base address                                                                                                                                          |
| <i>mask</i> | The interrupts to disable The parameter can be combination of the following source if defined <ul style="list-style-type: none"><li>• kEWM_InterruptEnable</li></ul> |

### 20.6.6 static uint32\_t EWM\_GetStatusFlags ( EWM\_Type \* *base* ) [inline], [static]

This function gets all status flags.

This is an example for getting the running flag.

```

* uint32_t status;
* status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
*

```

## Parameters

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

## Returns

State of the status flag: asserted (true) or not-asserted (false).

## See Also

[\\_ewm\\_status\\_flags\\_t](#)

- True: a related status flag has been set.
- False: a related status flag is not set.

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

This function resets the EWM counter to zero.

## Parameters

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





## Chapter 21

# FlexCAN: Flex Controller Area Network Driver

### 21.1 Overview

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

#### Modules

- [FlexCAN Driver](#)
- [FlexCAN eDMA Driver](#)

### 21.2 FlexCAN Driver

#### 21.2.1 Overview

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

#### 21.2.2 Typical use case

##### 21.2.2.1 Message Buffer Send Operation

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

##### 21.2.2.2 Message Buffer Receive Operation

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

##### 21.2.2.3 Receive FIFO Operation

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

### Data Structures

- struct `flexcan_frame_t`  
*FlexCAN message frame structure. [More...](#)*
- struct `flexcan_timing_config_t`  
*FlexCAN protocol timing characteristic configuration structure. [More...](#)*
- struct `flexcan_config_t`  
*FlexCAN module configuration structure. [More...](#)*
- struct `flexcan_rx_mb_config_t`  
*FlexCAN Receive Message Buffer configuration structure. [More...](#)*
- struct `flexcan_rx_fifo_config_t`  
*FlexCAN Rx FIFO configuration structure. [More...](#)*
- struct `flexcan_mb_transfer_t`  
*FlexCAN Message Buffer transfer. [More...](#)*
- struct `flexcan_fifo_transfer_t`  
*FlexCAN Rx FIFO transfer. [More...](#)*
- struct `flexcan_handle_t`  
*FlexCAN handle structure. [More...](#)*

## Macros

- #define `FLEXCAN_ID_STD(id)` (((uint32\_t)((uint32\_t)(id)) << CAN\_ID\_STD\_SHIFT) & CAN\_ID\_STD\_MASK)  
*FlexCAN Frame ID helper macro.*
- #define `FLEXCAN_ID_EXT(id)`  
*Extend Frame ID helper macro.*
- #define `FLEXCAN_RX_MB_STD_MASK(id, rtr, ide)`  
*FlexCAN Rx Message Buffer Mask helper macro.*
- #define `FLEXCAN_RX_MB_EXT_MASK(id, rtr, ide)`  
*Extend Rx Message Buffer Mask helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)`  
*FlexCAN Rx FIFO Mask helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(id, rtr, ide)`  
*Standard Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(id, rtr, ide)`  
*Standard Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id)` (((uint32\_t)(id)&0x7F8) << 21)  
*Standard Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id)` (((uint32\_t)(id)&0x7F8) << 13)  
*Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id)` (((uint32\_t)(id)&0x7F8) << 5)  
*Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)` (((uint32\_t)(id)&0x7F8) >> 3)  
*Standard Rx FIFO Mask helper macro Type C lower part helper macro.*
- #define `FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)`  
*Extend Rx FIFO Mask helper macro Type A helper macro.*
- #define `FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(id, rtr, ide)`  
*Extend Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define `FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(id, rtr, ide)`  
*Extend Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define `FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)` ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) << 3)  
*Extend Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define `FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(id)`  
*Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.*
- #define `FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)`  
*Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- #define `FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)` ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) >> 21)  
*Extend Rx FIFO Mask helper macro Type C lower part helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide)` `FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)`  
*FlexCAN Rx FIFO Filter helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)`  
*Standard Rx FIFO Filter helper macro Type B upper part helper macro.*
- #define `FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)`

## FlexCAN Driver

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

## Typedefs

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

## Enumerations

- enum `_flexcan_status` {  
`kStatus_FLEXCAN_TxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 0),  
`kStatus_FLEXCAN_TxIdle` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 1),  
`kStatus_FLEXCAN_TxSwitchToRx`,  
`kStatus_FLEXCAN_RxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 3),  
`kStatus_FLEXCAN_RxIdle` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 4),  
`kStatus_FLEXCAN_RxOverflow` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 5),  
`kStatus_FLEXCAN_RxFifoBusy` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 6),  
`kStatus_FLEXCAN_RxFifoIdle` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 7),  
`kStatus_FLEXCAN_RxFifoOverflow` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 8),  
`kStatus_FLEXCAN_RxFifoWarning` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 9),  
`kStatus_FLEXCAN_ErrorStatus` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 10),



```
kStatus_FLEXCAN_UnHandled = MAKE_STATUS(kStatusGroup_FLEXCAN, 11) }
```

*FlexCAN transfer status.*

- enum flexcan\_frame\_format\_t {  
 kFLEXCAN\_FrameFormatStandard = 0x0U,  
 kFLEXCAN\_FrameFormatExtend = 0x1U }

*FlexCAN frame format.*

- enum flexcan\_frame\_type\_t {  
 kFLEXCAN\_FrameTypeData = 0x0U,  
 kFLEXCAN\_FrameTypeRemote = 0x1U }

*FlexCAN frame type.*

- enum flexcan\_rx\_fifo\_filter\_type\_t {  
 kFLEXCAN\_RxFifoFilterTypeA = 0x0U,  
 kFLEXCAN\_RxFifoFilterTypeB,  
 kFLEXCAN\_RxFifoFilterTypeC,  
 kFLEXCAN\_RxFifoFilterTypeD = 0x3U }

*FlexCAN Rx Fifo Filter type.*

- enum flexcan\_rx\_fifo\_priority\_t {  
 kFLEXCAN\_RxFifoPrioLow = 0x0U,  
 kFLEXCAN\_RxFifoPrioHigh = 0x1U }

*FlexCAN Rx FIFO priority.*

- enum \_flexcan\_interrupt\_enable {  
 kFLEXCAN\_BusOffInterruptEnable = CAN\_CTRL1\_BOFFMSK\_MASK,  
 kFLEXCAN\_ErrorInterruptEnable = CAN\_CTRL1\_ERRMSK\_MASK,  
 kFLEXCAN\_RxWarningInterruptEnable = CAN\_CTRL1\_RWRNMSK\_MASK,  
 kFLEXCAN\_TxWarningInterruptEnable = CAN\_CTRL1\_TWRNMSK\_MASK,  
 kFLEXCAN\_WakeUpInterruptEnable = CAN\_MCR\_WAKMSK\_MASK }

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

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

*FlexCAN status flags.*

- enum \_flexcan\_error\_flags {

## FlexCAN Driver

```
kFLEXCAN_StuffingError = CAN_ESR1_STFERR_MASK,  
kFLEXCAN_FormError = CAN_ESR1_FRMERR_MASK,  
kFLEXCAN_CrcError = CAN_ESR1_CRCERR_MASK,  
kFLEXCAN_AckError = CAN_ESR1_ACKERR_MASK,  
kFLEXCAN_Bit0Error = CAN_ESR1_BIT0ERR_MASK,  
kFLEXCAN_Bit1Error = CAN_ESR1_BIT1ERR_MASK }
```

*FlexCAN error status flags.*

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

*FlexCAN Rx FIFO status flags.*

## Driver version

- #define `FSL_FLEXCAN_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 0))  
*FlexCAN driver version 2.2.0.*

## Initialization and deinitialization

- uint32\_t `FLEXCAN_GetInstance` (CAN\_Type \*base)  
*Get the FlexCAN instance from peripheral base address.*
- void `FLEXCAN_Init` (CAN\_Type \*base, const `flexcan_config_t` \*config, uint32\_t sourceClock\_Hz)  
*Initializes a FlexCAN instance.*
- void `FLEXCAN_Deinit` (CAN\_Type \*base)  
*De-initializes a FlexCAN instance.*
- void `FLEXCAN_GetDefaultConfig` (`flexcan_config_t` \*config)  
*Gets the default configuration structure.*

## Configuration.

- void `FLEXCAN_SetTimingConfig` (CAN\_Type \*base, const `flexcan_timing_config_t` \*config)  
*Sets the FlexCAN protocol timing characteristic.*
- void `FLEXCAN_SetRxMbGlobalMask` (CAN\_Type \*base, uint32\_t mask)  
*Sets the FlexCAN receive message buffer global mask.*
- void `FLEXCAN_SetRxFifoGlobalMask` (CAN\_Type \*base, uint32\_t mask)  
*Sets the FlexCAN receive FIFO global mask.*
- void `FLEXCAN_SetRxIndividualMask` (CAN\_Type \*base, uint8\_t maskIdx, uint32\_t mask)  
*Sets the FlexCAN receive individual mask.*
- void `FLEXCAN_SetTxMbConfig` (CAN\_Type \*base, uint8\_t mbIdx, bool enable)  
*Configures a FlexCAN transmit message buffer.*
- void `FLEXCAN_SetRxMbConfig` (CAN\_Type \*base, uint8\_t mbIdx, const `flexcan_rx_mb_config_t` \*config, bool enable)  
*Configures a FlexCAN Receive Message Buffer.*
- void `FLEXCAN_SetRxFifoConfig` (CAN\_Type \*base, const `flexcan_rx_fifo_config_t` \*config, bool enable)

*Configures the FlexCAN Rx FIFO.*

## Status

- static uint32\_t [FLEXCAN\\_GetStatusFlags](#) (CAN\_Type \*base)  
*Gets the FlexCAN module interrupt flags.*
- static void [FLEXCAN\\_ClearStatusFlags](#) (CAN\_Type \*base, uint32\_t mask)  
*Clears status flags with the provided mask.*
- static void [FLEXCAN\\_GetBusErrCount](#) (CAN\_Type \*base, uint8\_t \*txErrBuf, uint8\_t \*rxErrBuf)  
*Gets the FlexCAN Bus Error Counter value.*
- static uint64\_t [FLEXCAN\\_GetMbStatusFlags](#) (CAN\_Type \*base, uint64\_t mask)  
*Gets the FlexCAN Message Buffer interrupt flags.*
- static void [FLEXCAN\\_ClearMbStatusFlags](#) (CAN\_Type \*base, uint64\_t mask)  
*Clears the FlexCAN Message Buffer interrupt flags.*

## Interrupts

- static void [FLEXCAN\\_EnableInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Enables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_DisableInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Disables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_EnableMbInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Enables FlexCAN Message Buffer interrupts.*
- static void [FLEXCAN\\_DisableMbInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Disables FlexCAN Message Buffer interrupts.*

## Bus Operations

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

## Transactional

- status\_t [FLEXCAN\\_TransferSendBlocking](#) (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_frame\_t \*txFrame)  
*Performs a polling send transaction on the CAN bus.*
- status\_t [FLEXCAN\\_TransferReceiveBlocking](#) (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_frame\_t \*rxFrame)

## FlexCAN Driver

- Performs a polling receive transaction on the CAN bus.*
- status\_t **FLEXCAN\_TransferReceiveFifoBlocking** (CAN\_Type \*base, flexcan\_frame\_t \*rxFrame)  
*Performs a polling receive transaction from Rx FIFO on the CAN bus.*
- void **FLEXCAN\_TransferCreateHandle** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_transfer\_callback\_t callback, void \*userData)  
*Initializes the FlexCAN handle.*
- status\_t **FLEXCAN\_TransferSendNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*xfer)  
*Sends a message using IRQ.*
- status\_t **FLEXCAN\_TransferReceiveNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*xfer)  
*Receives a message using IRQ.*
- status\_t **FLEXCAN\_TransferReceiveFifoNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_fifo\_transfer\_t \*xfer)  
*Receives a message from Rx FIFO using IRQ.*
- void **FLEXCAN\_TransferAbortSend** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)  
*Aborts the interrupt driven message send process.*
- void **FLEXCAN\_TransferAbortReceive** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)  
*Aborts the interrupt driven message receive process.*
- void **FLEXCAN\_TransferAbortReceiveFifo** (CAN\_Type \*base, flexcan\_handle\_t \*handle)  
*Aborts the interrupt driven message receive from Rx FIFO process.*
- void **FLEXCAN\_TransferHandleIRQ** (CAN\_Type \*base, flexcan\_handle\_t \*handle)  
*FlexCAN IRQ handle function.*

## 21.2.3 Data Structure Documentation

### 21.2.3.1 struct flexcan\_frame\_t

#### 21.2.3.1.0.28 Field Documentation

21.2.3.1.0.28.1 uint32\_t flexcan\_frame\_t::timestamp

21.2.3.1.0.28.2 uint32\_t flexcan\_frame\_t::length

21.2.3.1.0.28.3 uint32\_t flexcan\_frame\_t::type

21.2.3.1.0.28.4 uint32\_t flexcan\_frame\_t::format

21.2.3.1.0.28.5 uint32\_t flexcan\_frame\_t::\_\_pad0\_\_

21.2.3.1.0.28.6 uint32\_t flexcan\_frame\_t::idhit

21.2.3.1.0.28.7 uint32\_t flexcan\_frame\_t::id

21.2.3.1.0.28.8 uint32\_t flexcan\_frame\_t::dataWord0

21.2.3.1.0.28.9 uint32\_t flexcan\_frame\_t::dataWord1

21.2.3.1.0.28.10 uint8\_t flexcan\_frame\_t::dataByte3

21.2.3.1.0.28.11 uint8\_t flexcan\_frame\_t::dataByte2

21.2.3.1.0.28.12 uint8\_t flexcan\_frame\_t::dataByte1

21.2.3.1.0.28.13 uint8\_t flexcan\_frame\_t::dataByte0

21.2.3.1.0.28.14 uint8\_t flexcan\_frame\_t::dataByte7

21.2.3.1.0.28.15 uint8\_t flexcan\_frame\_t::dataByte6

21.2.3.1.0.28.16 uint8\_t flexcan\_frame\_t::dataByte5

21.2.3.1.0.28.17 uint8\_t flexcan\_frame\_t::dataByte4

### 21.2.3.2 struct flexcan\_timing\_config\_t

#### Data Fields

- uint16\_t [preDivider](#)  
*Clock Pre-scaler Division Factor.*
- uint8\_t [rJumpwidth](#)  
*Re-sync Jump Width.*
- uint8\_t [phaseSeg1](#)  
*Phase Segment 1.*

## FlexCAN Driver

- uint8\_t [phaseSeg2](#)  
*Phase Segment 2.*
- uint8\_t [propSeg](#)  
*Propagation Segment.*

### 21.2.3.2.0.29 Field Documentation

21.2.3.2.0.29.1 uint16\_t flexcan\_timing\_config\_t::preDivider

21.2.3.2.0.29.2 uint8\_t flexcan\_timing\_config\_t::rJumpwidth

21.2.3.2.0.29.3 uint8\_t flexcan\_timing\_config\_t::phaseSeg1

21.2.3.2.0.29.4 uint8\_t flexcan\_timing\_config\_t::phaseSeg2

21.2.3.2.0.29.5 uint8\_t flexcan\_timing\_config\_t::propSeg

### 21.2.3.3 struct flexcan\_config\_t

#### Data Fields

- uint32\_t [baudRate](#)  
*FlexCAN baud rate in bps.*
- uint8\_t [maxMbNum](#)  
*The maximum number of Message Buffers used by user.*
- bool [enableLoopBack](#)  
*Enable or Disable Loop Back Self Test Mode.*
- bool [enableSelfWakeup](#)  
*Enable or Disable Self Wakeup Mode.*
- bool [enableIndividMask](#)  
*Enable or Disable Rx Individual Mask.*

### 21.2.3.3.0.30 Field Documentation

21.2.3.3.0.30.1 uint32\_t flexcan\_config\_t::baudRate

21.2.3.3.0.30.2 uint8\_t flexcan\_config\_t::maxMbNum

21.2.3.3.0.30.3 bool flexcan\_config\_t::enableLoopBack

21.2.3.3.0.30.4 bool flexcan\_config\_t::enableSelfWakeup

21.2.3.3.0.30.5 bool flexcan\_config\_t::enableIndividMask

### 21.2.3.4 struct flexcan\_rx\_mb\_config\_t

This structure is used as the parameter of [FLEXCAN\\_SetRxMbConfig\(\)](#) function. The [FLEXCAN\\_SetRxMbConfig\(\)](#) function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

## Data Fields

- `uint32_t id`  
*CAN Message Buffer Frame Identifier, should be set using `FLEXCAN_ID_EXT()` or `FLEXCAN_ID_STD()` macro.*
- `flexcan_frame_format_t format`  
*CAN Frame Identifier format(Standard of Extend).*
- `flexcan_frame_type_t type`  
*CAN Frame Type(Data or Remote).*

### 21.2.3.4.0.31 Field Documentation

21.2.3.4.0.31.1 `uint32_t flexcan_rx_mb_config_t::id`

21.2.3.4.0.31.2 `flexcan_frame_format_t flexcan_rx_mb_config_t::format`

21.2.3.4.0.31.3 `flexcan_frame_type_t flexcan_rx_mb_config_t::type`

### 21.2.3.5 struct flexcan\_rx\_fifo\_config\_t

## Data Fields

- `uint32_t * idFilterTable`  
*Pointer to the FlexCAN Rx FIFO identifier filter table.*
- `uint8_t idFilterNum`  
*The quantity of filter elements.*
- `flexcan_rx_fifo_filter_type_t idFilterType`  
*The FlexCAN Rx FIFO Filter type.*
- `flexcan_rx_fifo_priority_t priority`  
*The FlexCAN Rx FIFO receive priority.*

### 21.2.3.5.0.32 Field Documentation

21.2.3.5.0.32.1 `uint32_t* flexcan_rx_fifo_config_t::idFilterTable`

21.2.3.5.0.32.2 `uint8_t flexcan_rx_fifo_config_t::idFilterNum`

21.2.3.5.0.32.3 `flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType`

21.2.3.5.0.32.4 `flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority`

### 21.2.3.6 struct flexcan\_mb\_transfer\_t

## Data Fields

- `flexcan_frame_t * frame`  
*The buffer of CAN Message to be transfer.*
- `uint8_t mblIdx`  
*The index of Message buffer used to transfer Message.*

## FlexCAN Driver

### 21.2.3.6.0.33 Field Documentation

21.2.3.6.0.33.1 `flexcan_frame_t* flexcan_mb_transfer_t::frame`

21.2.3.6.0.33.2 `uint8_t flexcan_mb_transfer_t::mbldx`

### 21.2.3.7 struct `flexcan_fifo_transfer_t`

#### Data Fields

- `flexcan_frame_t * frame`  
*The buffer of CAN Message to be received from Rx FIFO.*

### 21.2.3.7.0.34 Field Documentation

21.2.3.7.0.34.1 `flexcan_frame_t* flexcan_fifo_transfer_t::frame`

### 21.2.3.8 struct `_flexcan_handle`

FlexCAN handle structure definition.

#### Data Fields

- `flexcan_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*FlexCAN callback function parameter.*
- `flexcan_frame_t *volatile mbFrameBuf [CAN_WORD1_COUNT]`  
*The buffer for received data from Message Buffers.*
- `flexcan_frame_t *volatile rxFifoFrameBuf`  
*The buffer for received data from Rx FIFO.*
- `volatile uint8_t mbState [CAN_WORD1_COUNT]`  
*Message Buffer transfer state.*
- `volatile uint8_t rxFifoState`  
*Rx FIFO transfer state.*



### 21.2.3.8.0.35 Field Documentation

21.2.3.8.0.35.1 `flexcan_transfer_callback_t flexcan_handle_t::callback`

21.2.3.8.0.35.2 `void* flexcan_handle_t::userData`

21.2.3.8.0.35.3 `flexcan_frame_t* volatile flexcan_handle_t::mbFrameBuf[CAN_WORD1_COUNT]`

21.2.3.8.0.35.4 `flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf`

21.2.3.8.0.35.5 `volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]`

21.2.3.8.0.35.6 `volatile uint8_t flexcan_handle_t::rxFifoState`

## 21.2.4 Macro Definition Documentation

21.2.4.1 `#define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

21.2.4.2 `#define FLEXCAN_ID_STD( id ) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)`

Standard Frame ID helper macro.

21.2.4.3 `#define FLEXCAN_ID_EXT( id )`

**Value:**

```
(((uint32_t)((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
(CAN_ID_EXT_MASK | CAN_ID_STD_MASK)
```

21.2.4.4 `#define FLEXCAN_RX_MB_STD_MASK( id, rtr, ide )`

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_STD(id))
```

Standard Rx Message Buffer Mask helper macro.

21.2.4.5 `#define FLEXCAN_RX_MB_EXT_MASK( id, rtr, ide )`

**Value:**

```
(((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_EXT(id))
```

## FlexCAN Driver

### 21.2.4.6 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )

#### Value:

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \  
 (FLEXCAN_ID_STD(id) << 1)
```

Standard Rx FIFO Mask helper macro Type A helper macro.

### 21.2.4.7 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )

#### Value:

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \  
 ((uint32_t)(id) & 0x7FF) << 19)
```

### 21.2.4.8 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )

#### Value:

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \  
 ((uint32_t)(id) & 0x7FF) << 3)
```

### 21.2.4.9 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH( *id* ) (((uint32\_t)(id) & 0x7F8) << 21)

### 21.2.4.10 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH( *id* ) (((uint32\_t)(id) & 0x7F8) << 13)

### 21.2.4.11 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW( *id* ) (((uint32\_t)(id) & 0x7F8) << 5)

### 21.2.4.12 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW( *id* ) (((uint32\_t)(id) & 0x7F8) >> 3)

### 21.2.4.13 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )

#### Value:

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \  
 (FLEXCAN_ID_EXT(id) << 1)
```

**21.2.4.14 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )****Value:**

```
(
  ((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
  ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) << 1))
```

**21.2.4.15 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
  ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >> 15))
```

**21.2.4.16 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) << 3)****21.2.4.17 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH( *id* )****Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 5)
```

**21.2.4.18 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW( *id* )****Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 13)
```

**21.2.4.19 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) >> 21)****21.2.4.20 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* ) FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A(id, rtr, ide)**

Standard Rx FIFO Filter helper macro Type A helper macro.

## FlexCAN Driver

**21.2.4.21 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(           \  
    id, rtr, ide)
```

**21.2.4.22 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(           \  
    id, rtr, ide)
```

**21.2.4.23 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(         \  
    id)
```

**21.2.4.24 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(     \  
    id)
```

**21.2.4.25 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(     \  
    id)
```

**21.2.4.26 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(         \  
    id)
```

\\\\\\\\\\

**21.2.4.27 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A(id, rtr, ide)**

**21.2.4.28 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(
    id, rtr, ide) \
```

**21.2.4.29 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(
    id, rtr, ide) \
```

**21.2.4.30 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(
    id) \
```

\\\\\\\\

**21.2.4.31 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(
    id) \
```

**21.2.4.32 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(
    id) \
```

```
21.2.4.33 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW( id
           ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)
```

### 21.2.5 Typedef Documentation

```
21.2.5.1 typedef void(* flexcan_transfer_callback_t)(CAN_Type *base, flexcan_handle_t
           *handle, status_t status, uint32_t result, void *userData)
```

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to `kStatus_FLEXCAN_ErrorStatus`, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

### 21.2.6 Enumeration Type Documentation

#### 21.2.6.1 enum flexcan\_status

Enumerator

*kStatus\_FLEXCAN\_TxBusy* Tx Message Buffer is Busy.  
*kStatus\_FLEXCAN\_TxIdle* Tx Message Buffer is Idle.  
*kStatus\_FLEXCAN\_TxSwitchToRx* Remote Message is send out and Message buffer changed to Receive one.  
*kStatus\_FLEXCAN\_RxBusy* Rx Message Buffer is Busy.  
*kStatus\_FLEXCAN\_RxIdle* Rx Message Buffer is Idle.  
*kStatus\_FLEXCAN\_RxOverflow* Rx Message Buffer is Overflowed.  
*kStatus\_FLEXCAN\_RxFifoBusy* Rx Message FIFO is Busy.  
*kStatus\_FLEXCAN\_RxFifoIdle* Rx Message FIFO is Idle.  
*kStatus\_FLEXCAN\_RxFifoOverflow* Rx Message FIFO is overflowed.  
*kStatus\_FLEXCAN\_RxFifoWarning* Rx Message FIFO is almost overflowed.  
*kStatus\_FLEXCAN\_ErrorStatus* FlexCAN Module Error and Status.  
*kStatus\_FLEXCAN\_UnHandled* UnHadled Interrupt asserted.

#### 21.2.6.2 enum flexcan\_frame\_format\_t

Enumerator

*kFLEXCAN\_FrameFormatStandard* Standard frame format attribute.  
*kFLEXCAN\_FrameFormatExtend* Extend frame format attribute.

### 21.2.6.3 enum flexcan\_frame\_type\_t

Enumerator

- kFLEXCAN\_FrameTypeData* Data frame type attribute.
- kFLEXCAN\_FrameTypeRemote* Remote frame type attribute.

### 21.2.6.4 enum flexcan\_rx\_fifo\_filter\_type\_t

Enumerator

- kFLEXCAN\_RxFifoFilterTypeA* One full ID (standard and extended) per ID Filter element.
- kFLEXCAN\_RxFifoFilterTypeB* Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.
- kFLEXCAN\_RxFifoFilterTypeC* Four partial 8-bit Standard or extended ID slices per ID Filter Table element.
- kFLEXCAN\_RxFifoFilterTypeD* All frames rejected.

### 21.2.6.5 enum flexcan\_rx\_fifo\_priority\_t

The matching process starts from the Rx MB(or Rx FIFO) with higher priority. If no MB(or Rx FIFO filter) is satisfied, the matching process goes on with the Rx FIFO(or Rx MB) with lower priority.

Enumerator

- kFLEXCAN\_RxFifoPrioLow* Matching process start from Rx Message Buffer first.
- kFLEXCAN\_RxFifoPrioHigh* Matching process start from Rx FIFO first.

### 21.2.6.6 enum \_flexcan\_interrupt\_enable

This structure contains the settings for all of the FlexCAN Module interrupt configurations. Note: FlexCAN Message Buffers and Rx FIFO have their own interrupts.

Enumerator

- kFLEXCAN\_BusOffInterruptEnable* Bus Off interrupt.
- kFLEXCAN\_ErrorInterruptEnable* Error interrupt.
- kFLEXCAN\_RxWarningInterruptEnable* Rx Warning interrupt.
- kFLEXCAN\_TxWarningInterruptEnable* Tx Warning interrupt.
- kFLEXCAN\_WakeUpInterruptEnable* Wake Up interrupt.

## FlexCAN Driver

### 21.2.6.7 enum `_flexcan_flags`

This provides constants for the FlexCAN status flags for use in the FlexCAN functions. Note: The CPU read action clears `FIEXCAN_ErrorFlag`, therefore user need to read `FIEXCAN_ErrorFlag` and distinguish which error is occur using `_flexcan_error_flags` enumerations.

Enumerator

***kFLEXCAN\_SynchFlag*** CAN Synchronization Status.  
***kFLEXCAN\_TxWarningIntFlag*** Tx Warning Interrupt Flag.  
***kFLEXCAN\_RxWarningIntFlag*** Rx Warning Interrupt Flag.  
***kFLEXCAN\_TxErrorWarningFlag*** Tx Error Warning Status.  
***kFLEXCAN\_RxErrorWarningFlag*** Rx Error Warning Status.  
***kFLEXCAN\_IdleFlag*** CAN IDLE Status Flag.  
***kFLEXCAN\_FaultConfinementFlag*** Fault Confinement State Flag.  
***kFLEXCAN\_TransmittingFlag*** FlexCAN In Transmission Status.  
***kFLEXCAN\_ReceivingFlag*** FlexCAN In Reception Status.  
***kFLEXCAN\_BusOffIntFlag*** Bus Off Interrupt Flag.  
***kFLEXCAN\_ErrorIntFlag*** Error Interrupt Flag.  
***kFLEXCAN\_WakeUpIntFlag*** Wake-Up Interrupt Flag.  
***kFLEXCAN\_ErrorFlag*** All FlexCAN Error Status.

### 21.2.6.8 enum `_flexcan_error_flags`

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with `KFLEXCAN_ErrorFlag` in `_flexcan_flags` enumerations to determine which error is generated.

Enumerator

***kFLEXCAN\_StuffingError*** Stuffing Error.  
***kFLEXCAN\_FormError*** Form Error.  
***kFLEXCAN\_CrcError*** Cyclic Redundancy Check Error.  
***kFLEXCAN\_AckError*** Received no ACK on transmission.  
***kFLEXCAN\_Bit0Error*** Unable to send dominant bit.  
***kFLEXCAN\_Bit1Error*** Unable to send recessive bit.

### 21.2.6.9 enum `_flexcan_rx_fifo_flags`

The FlexCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Enumerator

***kFLEXCAN\_RxFifoOverflowFlag*** Rx FIFO overflow flag.



*kFLEXCAN\_RxFifoWarningFlag* Rx FIFO almost full flag.  
*kFLEXCAN\_RxFifoFrameAvlFlag* Frames available in Rx FIFO flag.

## 21.2.7 Function Documentation

### 21.2.7.1 uint32\_t FLEXCAN\_GetInstance ( CAN\_Type \* *base* )

Parameters

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

Returns

FlexCAN instance.

### 21.2.7.2 void FLEXCAN\_Init ( CAN\_Type \* *base*, const flexcan\_config\_t \* *config*, uint32\_t *sourceClock\_Hz* )

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the `flexcan_config_t` parameters and how to call the `FLEXCAN_Init` function by passing in these parameters.

```
* flexcan_config_t flexcanConfig;
* flexcanConfig.clkSrc          = kFLEXCAN_ClkSrcOsc;
* flexcanConfig.baudRate       = 1000000U;
* flexcanConfig.maxMbNum       = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeup = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.enableDoze      = false;
* flexcanConfig.timingConfig    = timingConfig;
* FLEXCAN_Init(CAN0, &flexcanConfig, 8000000UL);
*
```

Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                      |
| <i>config</i>         | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |

### 21.2.7.3 void FLEXCAN\_Deinit ( CAN\_Type \* *base* )

This function disables the FlexCAN module clock and sets all register values to the reset value.

## FlexCAN Driver

### Parameters

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

### 21.2.7.4 void FLEXCAN\_GetDefaultConfig ( flexcan\_config\_t \* config )

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = KFLEXCAN\_ClkSrcOsc; flexcanConfig->baudRate = 1000000-U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->enableDoze = false; flexcanConfig.timingConfig = timingConfig;

### Parameters

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

### 21.2.7.5 void FLEXCAN\_SetTimingConfig ( CAN\_Type \* base, const flexcan\_timing\_config\_t \* config )

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN\\_Init\(\)](#) and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling [FLEXCAN\\_SetTimingConfig\(\)](#) overrides the baud rate set in [FLEXCAN\\_Init\(\)](#).

### Parameters

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

### 21.2.7.6 void FLEXCAN\_SetRxMbGlobalMask ( CAN\_Type \* base, uint32\_t mask )

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the [FLEXCAN\\_Init\(\)](#).

### Parameters

---

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.     |
| <i>mask</i> | Rx Message Buffer Global Mask value. |

### 21.2.7.7 void FLEXCAN\_SetRxFifoGlobalMask ( CAN\_Type \* *base*, uint32\_t *mask* )

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FlexCAN peripheral base address. |
| <i>mask</i> | Rx Fifo Global Mask value.       |

### 21.2.7.8 void FLEXCAN\_SetRxIndividualMask ( CAN\_Type \* *base*, uint8\_t *maskIdx*, uint32\_t *mask* )

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the [FLEXCAN\\_Init\(\)](#). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>base</i>    | FlexCAN peripheral base address. |
| <i>maskIdx</i> | The Index of individual Mask.    |
| <i>mask</i>    | Rx Individual Mask value.        |

### 21.2.7.9 void FLEXCAN\_SetTxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

## FlexCAN Driver

### Parameters

|               |                                                                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                       |
| <i>enable</i> | Enable/disable Tx Message Buffer. <ul style="list-style-type: none"><li>• true: Enable Tx Message Buffer.</li><li>• false: Disable Tx Message Buffer.</li></ul> |

### 21.2.7.10 void FLEXCAN\_SetRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_rx\_mb\_config\_t \* *config*, bool *enable* )

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

### Parameters

|               |                                                                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                       |
| <i>config</i> | Pointer to the FlexCAN Message Buffer configuration structure.                                                                                                  |
| <i>enable</i> | Enable/disable Rx Message Buffer. <ul style="list-style-type: none"><li>• true: Enable Rx Message Buffer.</li><li>• false: Disable Rx Message Buffer.</li></ul> |

### 21.2.7.11 void FLEXCAN\_SetRxFifoConfig ( CAN\_Type \* *base*, const flexcan\_rx\_fifo\_config\_t \* *config*, bool *enable* )

This function configures the Rx FIFO with given Rx FIFO configuration.

### Parameters

|               |                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                  |
| <i>config</i> | Pointer to the FlexCAN Rx FIFO configuration structure.                                                                           |
| <i>enable</i> | Enable/disable Rx FIFO. <ul style="list-style-type: none"><li>• true: Enable Rx FIFO.</li><li>• false: Disable Rx FIFO.</li></ul> |

### 21.2.7.12 `static uint32_t FLEXCAN_GetStatusFlags ( CAN_Type * base ) [inline], [static]`

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators `_flexcan_flags`. To check the specific status, compare the return value with enumerators in `_flexcan_flags`.

Parameters

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

Returns

FlexCAN status flags which are ORed by the enumerators in the `_flexcan_flags`.

### 21.2.7.13 `static void FLEXCAN_ClearStatusFlags ( CAN_Type * base, uint32_t mask ) [inline], [static]`

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

|             |                                                                                         |
|-------------|-----------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                        |
| <i>mask</i> | The status flags to be cleared, it is logical OR value of <code>_flexcan_flags</code> . |

### 21.2.7.14 `static void FLEXCAN_GetBusErrCount ( CAN_Type * base, uint8_t * txErrBuf, uint8_t * rxErrBuf ) [inline], [static]`

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>txErrBuf</i> | Buffer to store Tx Error Counter value. |
| <i>rxErrBuf</i> | Buffer to store Rx Error Counter value. |

### 21.2.7.15 `static uint64_t FLEXCAN_GetMbStatusFlags ( CAN_Type * base, uint64_t mask ) [inline], [static]`

This function gets the interrupt flags of a given Message Buffers.

## FlexCAN Driver

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

### Returns

The status of given Message Buffers.

**21.2.7.16 static void FLEXCAN\_ClearMbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* )  
[inline], [static]**

This function clears the interrupt flags of a given Message Buffers.

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

**21.2.7.17 static void FLEXCAN\_EnableInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

### Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

**21.2.7.18 static void FLEXCAN\_DisableInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                  |
| <i>mask</i> | The interrupts to disable. Logical OR of <code>_flexcan_interrupt_enable</code> . |

**21.2.7.19** `static void FLEXCAN_EnableMblInterrupts ( CAN_Type * base, uint64_t mask ) [inline], [static]`

This function enables the interrupts of given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

**21.2.7.20** `static void FLEXCAN_DisableMblInterrupts ( CAN_Type * base, uint64_t mask ) [inline], [static]`

This function disables the interrupts of given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

**21.2.7.21** `static void FLEXCAN_Enable ( CAN_Type * base, bool enable ) [inline], [static]`

This function enables or disables the FlexCAN module.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN base pointer.             |
| <i>enable</i> | true to enable, false to disable. |

**21.2.7.22** `status_t FLEXCAN_WriteTxMb ( CAN_Type * base, uint8_t mbIdx, const flexcan_frame_t * txFrame )`

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

## FlexCAN Driver

### Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.         |
| <i>mbIdx</i>   | The FlexCAN Message Buffer index.        |
| <i>txFrame</i> | Pointer to CAN message frame to be sent. |

### Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

### 21.2.7.23 **status\_t FLEXCAN\_ReadRxMb ( CAN\_Type \* base, uint8\_t mbIdx, flexcan\_frame\_t \* rxFrame )**

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

### Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                      |
| <i>mbIdx</i>   | The FlexCAN Message Buffer index.                     |
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |

### Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 21.2.7.24 **status\_t FLEXCAN\_ReadRxFifo ( CAN\_Type \* base, flexcan\_frame\_t \* rxFrame )**

This function reads a CAN message from the FlexCAN build-in Rx FIFO.



## Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                      |
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |

## Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

### 21.2.7.25 **status\_t FLEXCAN\_TransferSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *txFrame* )**

Note that a transfer handle does not need to be created before calling this API.

## Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base pointer.         |
| <i>mbIdx</i>   | The FlexCAN Message Buffer index.        |
| <i>txFrame</i> | Pointer to CAN message frame to be sent. |

## Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

### 21.2.7.26 **status\_t FLEXCAN\_TransferReceiveBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *rxFrame* )**

Note that a transfer handle does not need to be created before calling this API.

## Parameters

|              |                                   |
|--------------|-----------------------------------|
| <i>base</i>  | FlexCAN peripheral base pointer.  |
| <i>mbIdx</i> | The FlexCAN Message Buffer index. |

## FlexCAN Driver

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |
|----------------|-------------------------------------------------------|

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 21.2.7.27 **status\_t FLEXCAN\_TransferReceiveFifoBlocking ( CAN\_Type \* *base*, flexcan\_frame\_t \* *rxFrame* )**

Note that a transfer handle does not need to be created before calling this API.

Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base pointer.                      |
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

### 21.2.7.28 **void FLEXCAN\_TransferCreateHandle ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_transfer\_callback\_t *callback*, void \* *userData* )**

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

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>handle</i>   | FlexCAN handle pointer.                 |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

**21.2.7.29** `status_t FLEXCAN_TransferSendNonBlocking ( CAN_Type * base,  
flexcan_handle_t * handle, flexcan_mb_transfer_t * xfer )`

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

## FlexCAN Driver

### Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                           |
| <i>handle</i> | FlexCAN handle pointer.                                                                    |
| <i>xfer</i>   | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

### Return values

|                                |                                                       |
|--------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>         | Start Tx Message Buffer sending process successfully. |
| <i>kStatus_Fail</i>            | Write Tx Message Buffer failed.                       |
| <i>kStatus_FLEXCAN_Tx-Busy</i> | Tx Message Buffer is in use.                          |

### 21.2.7.30 **status\_t FLEXCAN\_TransferReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *xfer* )**

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

### Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                           |
| <i>handle</i> | FlexCAN handle pointer.                                                                    |
| <i>xfer</i>   | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

### Return values

|                                |                                                           |
|--------------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>         | - Start Rx Message Buffer receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-Busy</i> | - Rx Message Buffer is in use.                            |

### 21.2.7.31 **status\_t FLEXCAN\_TransferReceiveFifoNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_fifo\_transfer\_t \* *xfer* )**

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

## Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                      |
| <i>handle</i> | FlexCAN handle pointer.                                                               |
| <i>xfer</i>   | FlexCAN Rx FIFO transfer structure. See the <a href="#">flexcan_fifo_transfer_t</a> . |

## Return values

|                                    |                                                 |
|------------------------------------|-------------------------------------------------|
| <i>kStatus_Success</i>             | - Start Rx FIFO receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | - Rx FIFO is currently in use.                  |

### 21.2.7.32 void FLEXCAN\_TransferAbortSend ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mblDx* )

This function aborts the interrupt driven message send process.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mblDx</i>  | The FlexCAN Message Buffer index. |

### 21.2.7.33 void FLEXCAN\_TransferAbortReceive ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mblDx* )

This function aborts the interrupt driven message receive process.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mblDx</i>  | The FlexCAN Message Buffer index. |

### 21.2.7.34 void FLEXCAN\_TransferAbortReceiveFifo ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function aborts the interrupt driven message receive from Rx FIFO process.

## FlexCAN Driver

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address. |
| <i>handle</i> | FlexCAN handle pointer.          |

### 21.2.7.35 void FLEXCAN\_TransferHandleIRQ ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address. |
| <i>handle</i> | FlexCAN handle pointer.          |

## 21.3 FlexCAN eDMA Driver

### 21.3.1 Overview

#### Data Structures

- struct [flexcan\\_edma\\_handle\\_t](#)  
*FlexCAN eDMA handle. [More...](#)*

#### Typedefs

- typedef void(\* [flexcan\\_edma\\_transfer\\_callback\\_t](#) )(CAN\_Type \*base, flexcan\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*FlexCAN transfer callback function.*

#### Driver version

- #define [FSL\\_FLEXCAN\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 0))  
*FlexCAN EDMA driver version 2.2.0.*

#### eDMA transactional

- void [FLEXCAN\\_TransferCreateHandleEDMA](#) (CAN\_Type \*base, flexcan\_edma\_handle\_t \*handle, [flexcan\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*rxFifoEdmaHandle)  
*Initializes the FlexCAN handle, which is used in transactional functions.*
- status\_t [FLEXCAN\\_TransferReceiveFifoEDMA](#) (CAN\_Type \*base, flexcan\_edma\_handle\_t \*handle, [flexcan\\_fifo\\_transfer\\_t](#) \*xfer)  
*Receives the CAN Message from the Rx FIFO using eDMA.*
- void [FLEXCAN\\_TransferAbortReceiveFifoEDMA](#) (CAN\_Type \*base, flexcan\_edma\_handle\_t \*handle)  
*Aborts the receive process which used eDMA.*

### 21.3.2 Data Structure Documentation

#### 21.3.2.1 struct [\\_flexcan\\_edma\\_handle](#)

##### Data Fields

- [flexcan\\_edma\\_transfer\\_callback\\_t](#) callback  
*Callback function.*
- void \* [userData](#)  
*FlexCAN callback function parameter.*
- [edma\\_handle\\_t](#) \* [rxFifoEdmaHandle](#)

## FlexCAN eDMA Driver

*The EDMA Rx FIFO channel used.*

- volatile uint8\_t **rxFifoState**  
*Rx FIFO transfer state.*

### 21.3.2.1.0.36 Field Documentation

21.3.2.1.0.36.1 flexcan\_edma\_transfer\_callback\_t flexcan\_edma\_handle\_t::callback

21.3.2.1.0.36.2 void\* flexcan\_edma\_handle\_t::userData

21.3.2.1.0.36.3 edma\_handle\_t\* flexcan\_edma\_handle\_t::rxFifoEdmaHandle

21.3.2.1.0.36.4 volatile uint8\_t flexcan\_edma\_handle\_t::rxFifoState

### 21.3.3 Macro Definition Documentation

21.3.3.1 #define FSL\_FLEXCAN\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

### 21.3.4 Typedef Documentation

21.3.4.1 typedef void(\* flexcan\_edma\_transfer\_callback\_t)(CAN\_Type \*base, flexcan\_edma\_handle\_t \*handle, status\_t status, void \*userData)

### 21.3.5 Function Documentation

21.3.5.1 void FLEXCAN\_TransferCreateHandleEDMA ( CAN\_Type \* base, flexcan\_edma\_handle\_t \* handle, flexcan\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* rxFifoEdmaHandle )

Parameters

|                          |                                                     |
|--------------------------|-----------------------------------------------------|
| <i>base</i>              | FlexCAN peripheral base address.                    |
| <i>handle</i>            | Pointer to flexcan_edma_handle_t structure.         |
| <i>callback</i>          | The callback function.                              |
| <i>userData</i>          | The parameter of the callback function.             |
| <i>rxFifoEdma-Handle</i> | User-requested DMA handle for Rx FIFO DMA transfer. |

21.3.5.2 status\_t FLEXCAN\_TransferReceiveFifoEDMA ( CAN\_Type \* base, flexcan\_edma\_handle\_t \* handle, flexcan\_fifo\_transfer\_t \* xfer )

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.



## Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                       |
| <i>handle</i> | Pointer to flexcan_edma_handle_t structure.                                            |
| <i>xfer</i>   | FlexCAN Rx FIFO EDMA transfer structure, see <a href="#">flexcan_fifo_transfer_t</a> . |

## Return values

|                                    |                            |
|------------------------------------|----------------------------|
| <i>kStatus_Success</i>             | if succeed, others failed. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | Previous transfer ongoing. |

### 21.3.5.3 void FLEXCAN\_TransferAbortReceiveFifoEDMA ( CAN\_Type \* *base*, flexcan\_edma\_handle\_t \* *handle* )

This function aborts the receive process which used eDMA.

## Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.            |
| <i>handle</i> | Pointer to flexcan_edma_handle_t structure. |



## Chapter 22

# FLEXRAM: on-chip RAM manager

### 22.1 Overview

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

The FLEXRAM module integrates the ITCM, DTCM, and OCRM controller, and supports parameterized RAM array and RAM array portioning.

This example code shows how to allocate RAM using the FLEXRAM driver.

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

### Data Structures

- struct `flexram_allocate_ram_t`  
*FLEXRAM allocate ocram, itcm, dtcm size. [More...](#)*

### Enumerations

- enum `_flexram_wr_rd_sel` {  
    `kFLEXRAM_Read` = 0U,  
    `kFLEXRAM_Write` = 1U }  
*flexram write read sel*
- enum `_flexram_interrupt_status` {  
    `kFLEXRAM_OCRAMAccessError` = FLEXRAM\_INT\_STATUS\_OCRAM\_ERR\_STATUS\_MASK,  
    `kFLEXRAM_DTCMAccessError` = FLEXRAM\_INT\_STATUS\_DTCM\_ERR\_STATUS\_MASK,  
    `kFLEXRAM_ITCMAccessError` = FLEXRAM\_INT\_STATUS\_ITCM\_ERR\_STATUS\_MASK,  
    `kFLEXRAM_OCRAMMagicAddrMatch` = FLEXRAM\_INT\_STATUS\_OCRAM\_MAM\_STATUS\_MASK,  
    `kFLEXRAM_DTCMMagicAddrMatch` = FLEXRAM\_INT\_STATUS\_DTCM\_MAM\_STATUS\_MASK,  
    `kFLEXRAM_ITCMMagicAddrMatch` = FLEXRAM\_INT\_STATUS\_ITCM\_MAM\_STATUS\_MASK,  
    `kFLEXRAM_InterruptStatusAll` = 0x3FU }  
*Interrupt status flag mask.*
- enum `flexram_tcm_access_mode_t` {  
    `kFLEXRAM_TCMAccessFastMode` = 0U,  
    `kFLEXRAM_TCMAccessWaitMode` = 1U }  
*FLEXRAM TCM access mode Fast access mode expected to be finished in 1-cycle Wait access mode expected to be finished in 2-cycle Wait access mode is a feature of the flexram and it should be used when the cpu clock too fast to finish tcm access in 1-cycle.*
- enum `_flexram_bank_type` {

## Overview

```
kFLEXRAM_BankNotUsed = 0U,  
kFLEXRAM_BankOCRAM = 1U,  
kFLEXRAM_BankDTCM = 2U,  
kFLEXRAM_BankITCM = 3U }
```

*FLEXRAM bank type.*

- enum `_flexram_tcm_size` {  
    kFLEXRAM\_TCMSize32KB = 32 \* 1024U,  
    kFLEXRAM\_TCMSize64KB = 64 \* 1024U,  
    kFLEXRAM\_TCMSize128KB = 128 \* 1024U,  
    kFLEXRAM\_TCMSize256KB = 256 \* 1024U,  
    kFLEXRAM\_TCMSize512KB = 512 \* 1024U }

*FLEXRAM tcm support size.*

- enum `flexram_bank_allocate_src_t` {  
    kFLEXRAM\_BankAllocateThroughHardwareFuse = 0U,  
    kFLEXRAM\_BankAllocateThroughBankCfg = 1U }

*FLEXRAM bank allocate source.*

## Variables

- const uint8\_t `flexram_allocate_ram_t::ocramBankNum`  
    *ocram banknumber which the SOC support*
- const uint8\_t `flexram_allocate_ram_t::dctmBankNum`  
    *dctm bank number to allocate, the number should be power of 2*
- const uint8\_t `flexram_allocate_ram_t::itcmBankNum`  
    *itcm bank number to allocate, the number should be power of 2*

## Driver version

- #define `FSL_FLEXRAM_DRIVER_VERSION` (MAKE\_VERSION(2U, 0U, 2U))  
    *Driver version 2.0.2.*

## Initialization and deinitialization

- void `FLEXRAM_Init` (FLEXRAM\_Type \*base)  
    *FLEXRAM module initialization function.*
- void `FLEXRAM_Deinit` (FLEXRAM\_Type \*base)  
    *Deinitializes the FLEXRAM.*

## Status

- static uint32\_t `FLEXRAM_GetInterruptStatus` (FLEXRAM\_Type \*base)  
    *FLEXRAM module get interrupt status.*
- static void `FLEXRAM_ClearInterruptStatus` (FLEXRAM\_Type \*base, uint32\_t status)  
    *FLEXRAM module clear interrupt status.*
- static void `FLEXRAM_EnableInterruptStatus` (FLEXRAM\_Type \*base, uint32\_t status)  
    *FLEXRAM module enable interrupt status.*
- static void `FLEXRAM_DisableInterruptStatus` (FLEXRAM\_Type \*base, uint32\_t status)  
    *FLEXRAM module disable interrupt status.*

## Interrupts

- static void [FLEXRAM\\_EnableInterruptSignal](#) (FLEXRAM\_Type \*base, uint32\_t status)  
*FLEXRAM module enable interrupt.*
- static void [FLEXRAM\\_DisableInterruptSignal](#) (FLEXRAM\_Type \*base, uint32\_t status)  
*FLEXRAM module disable interrupt.*

## functional

- static void [FLEXRAM\\_SetTCMReadAccessMode](#) (FLEXRAM\_Type \*base, flexram\_tcm\_access\_mode\_t mode)  
*FLEXRAM module set TCM read access mode.*
- static void [FLEXRAM\\_SetTCMWriteAccessMode](#) (FLEXRAM\_Type \*base, flexram\_tcm\_access\_mode\_t mode)  
*FLEXRAM module set TCM write access mode.*
- static void [FLEXRAM\\_EnableForceRamClockOn](#) (FLEXRAM\_Type \*base, bool enable)  
*FLEXRAM module force ram clock on.*
- static void [FLEXRAM\\_SetOCRAMMagicAddr](#) (FLEXRAM\_Type \*base, uint16\_t magicAddr, uint32\_t rwSel)  
*FLEXRAM OCRAM magic addr configuration When read/write access hit magic address, it will generate interrupt.*
- static void [FLEXRAM\\_SetDTCMMagicAddr](#) (FLEXRAM\_Type \*base, uint16\_t magicAddr, uint32\_t rwSel)  
*FLEXRAM DTCM magic addr configuration When read/write access hit magic address, it will generate interrupt.*
- static void [FLEXRAM\\_SetITCMMagicAddr](#) (FLEXRAM\_Type \*base, uint16\_t magicAddr, uint32\_t rwSel)  
*FLEXRAM ITCM magic addr configuration When read/write access hit magic address, it will generate interrupt.*
- status\_t [FLEXRAM\\_AllocateRam](#) (flexram\_allocate\_ram\_t \*config)  
*FLEXRAM allocate on-chip ram for OCRAM,ITCM,DTCM This function is independent of FLEXRAM\_Init, it can be called directly if ram re-allocate is needed.*
- static void [FLEXRAM\\_SetAllocateRamSrc](#) (flexram\_bank\_allocate\_src\_t src)  
*FLEXRAM set allocate on-chip ram source.*

## 22.2 Data Structure Documentation

### 22.2.1 struct flexram\_allocate\_ram\_t

#### Data Fields

- const uint8\_t [ocramBankNum](#)  
*ocram banknumber which the SOC support*
- const uint8\_t [dtdcmBankNum](#)  
*dtdcm bank number to allocate, the number should be power of 2*
- const uint8\_t [itcmBankNum](#)  
*itcm bank number to allocate, the number should be power of 2*

## Enumeration Type Documentation

### 22.3 Macro Definition Documentation

#### 22.3.1 #define FSL\_FLEXRAM\_DRIVER\_VERSION (MAKE\_VERSION(2U, 0U, 2U))

### 22.4 Enumeration Type Documentation

#### 22.4.1 enum \_flexram\_wr\_rd\_sel

Enumerator

*kFLEXRAM\_Read* read  
*kFLEXRAM\_Write* write

#### 22.4.2 enum \_flexram\_interrupt\_status

Enumerator

*kFLEXRAM\_OCRAMAccessError* ocram access unallocated address  
*kFLEXRAM\_DTCMAccessError* dtcm access unallocated address  
*kFLEXRAM\_ITCMAccessError* itcm access unallocated address  
*kFLEXRAM\_OCRAMMagicAddrMatch* ocram maigc address match  
*kFLEXRAM\_DTCMMagicAddrMatch* dtcm maigc address match  
*kFLEXRAM\_ITCMMagicAddrMatch* itcm maigc address match  
*kFLEXRAM\_InterruptStatusAll* all the interrupt status mask

#### 22.4.3 enum flexram\_tcm\_access\_mode\_t

Normally, fast mode is the default mode, the efficiency of the tcm access will better.

Enumerator

*kFLEXRAM\_TCMAccessFastMode* fast access mode  
*kFLEXRAM\_TCMAccessWaitMode* wait access mode

#### 22.4.4 enum \_flexram\_bank\_type

Enumerator

*kFLEXRAM\_BankNotUsed* bank is not used  
*kFLEXRAM\_BankOCRAM* bank is OCRAM  
*kFLEXRAM\_BankDTCM* bank is DTCM  
*kFLEXRAM\_BankITCM* bank is ITCM

## 22.4.5 enum `_flexram_tcm_size`

Enumerator

***kFLEXRAM\_TCMSize32KB*** TCM total size 32KB.  
***kFLEXRAM\_TCMSize64KB*** TCM total size 64KB.  
***kFLEXRAM\_TCMSize128KB*** TCM total size 128KB.  
***kFLEXRAM\_TCMSize256KB*** TCM total size 256KB.  
***kFLEXRAM\_TCMSize512KB*** TCM total size 512KB.

## 22.4.6 enum `flexram_bank_allocate_src_t`

Enumerator

***kFLEXRAM\_BankAllocateThroughHardwareFuse*** allocate ram through hardware fuse value  
***kFLEXRAM\_BankAllocateThroughBankCfg*** allocate ram through FLEXRAM\_BANK\_CFG

## 22.5 Function Documentation

### 22.5.1 void `FLEXRAM_Init ( FLEXRAM_Type * base )`

Parameters

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

### 22.5.2 static uint32\_t `FLEXRAM_GetInterruptStatus ( FLEXRAM_Type * base )` **`[inline], [static]`**

Parameters

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

### 22.5.3 static void `FLEXRAM_ClearInterruptStatus ( FLEXRAM_Type * base, uint32_t status )` **`[inline], [static]`**

## Function Documentation

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXRAM base address. |
| <i>status</i> | status to clear.      |

**22.5.4 static void FLEXRAM\_EnableInterruptStatus ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXRAM base address. |
| <i>status</i> | status to enable.     |

**22.5.5 static void FLEXRAM\_DisableInterruptStatus ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXRAM base address. |
| <i>status</i> | status to disable.    |

**22.5.6 static void FLEXRAM\_EnableInterruptSignal ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | FLEXRAM base address.       |
| <i>status</i> | status interrupt to enable. |

**22.5.7 static void FLEXRAM\_DisableInterruptSignal ( FLEXRAM\_Type \* *base*,  
uint32\_t *status* ) [inline], [static]**



Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | FLEXRAM base address.        |
| <i>status</i> | status interrupt to disable. |

**22.5.8 static void FLEXRAM\_SetTCMReadAccessMode ( FLEXRAM\_Type \* *base*, flexram\_tcm\_access\_mode\_t *mode* ) [inline], [static]**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXRAM base address. |
| <i>mode</i> | access mode.          |

**22.5.9 static void FLEXRAM\_SetTCMWriteAccessMode ( FLEXRAM\_Type \* *base*, flexram\_tcm\_access\_mode\_t *mode* ) [inline], [static]**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXRAM base address. |
| <i>mode</i> | access mode.          |

**22.5.10 static void FLEXRAM\_EnableForceRamClockOn ( FLEXRAM\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FLEXRAM base address.             |
| <i>enable</i> | enable or disable clock force on. |

**22.5.11 static void FLEXRAM\_SetOCRAMMagicAddr ( FLEXRAM\_Type \* *base*, uint16\_t *magicAddr*, uint32\_t *rwSel* ) [inline], [static]**

## Function Documentation

### Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>magicAddr</i> | magic address.                                    |
| <i>rwsel</i>     | read write select, 0 read access , 1 write access |

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

### Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>magicAddr</i> | magic address.                                    |
| <i>rwsel</i>     | read write select, 0 read access , 1 write access |

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

### Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>magicAddr</i> | magic address.                                    |
| <i>rwsel</i>     | read write select, 0 read access , 1 write access |

**22.5.14** `status_t FLEXRAM_AllocateRam ( flexram_allocate_ram_t * config )`

### Parameters

|               |                         |
|---------------|-------------------------|
| <i>config</i> | allocate configuration. |
|---------------|-------------------------|

### Return values

|                                |                         |
|--------------------------------|-------------------------|
| <i>kStatus_InvalidArgument</i> | the argument is invalid |
| <i>kStatus_Success</i>         | allocate success        |

**22.5.15** `static void FLEXRAM_SetAllocateRamSrc ( flexram_bank_allocate_src_t  
src ) [inline], [static]`

Parameters

|            |                                  |
|------------|----------------------------------|
| <i>src</i> | bank config source select value. |
|------------|----------------------------------|





## Chapter 23

### FlexIO: FlexIO Driver

#### 23.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FlexIO drivers for the FlexIO module of MCUXpresso SDK devices.

#### Modules

- [FlexIO Camera Driver](#)
- [FlexIO Driver](#)
- [FlexIO I2C Master Driver](#)
- [FlexIO I2S Driver](#)
- [FlexIO MCU Interface LCD Driver](#)
- [FlexIO SPI Driver](#)
- [FlexIO UART Driver](#)

### 23.2 FlexIO Driver

#### 23.2.1 Overview

#### Data Structures

- struct `flexio_config_t`  
*Define FlexIO user configuration structure. [More...](#)*
- struct `flexio_timer_config_t`  
*Define FlexIO timer configuration structure. [More...](#)*
- struct `flexio_shifter_config_t`  
*Define FlexIO shifter configuration structure. [More...](#)*

#### Macros

- #define `FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)` `((uint32_t)(x) << 1U)`  
*Calculate FlexIO timer trigger.*

#### Typedefs

- typedef void(\* `flexio_isr_t`)(void \*base, void \*handle)  
*typedef for FlexIO simulated driver interrupt handler.*

#### Enumerations

- enum `flexio_timer_trigger_polarity_t` {  
`kFLEXIO_TimerTriggerPolarityActiveHigh` = 0x0U,  
`kFLEXIO_TimerTriggerPolarityActiveLow` = 0x1U }  
*Define time of timer trigger polarity.*
- enum `flexio_timer_trigger_source_t` {  
`kFLEXIO_TimerTriggerSourceExternal` = 0x0U,  
`kFLEXIO_TimerTriggerSourceInternal` = 0x1U }  
*Define type of timer trigger source.*
- enum `flexio_pin_config_t` {  
`kFLEXIO_PinConfigOutputDisabled` = 0x0U,  
`kFLEXIO_PinConfigOpenDrainOrBidirection` = 0x1U,  
`kFLEXIO_PinConfigBidirectionOutputData` = 0x2U,  
`kFLEXIO_PinConfigOutput` = 0x3U }  
*Define type of timer/shifter pin configuration.*
- enum `flexio_pin_polarity_t` {  
`kFLEXIO_PinActiveHigh` = 0x0U,  
`kFLEXIO_PinActiveLow` = 0x1U }  
*Definition of pin polarity.*

- enum flexio\_timer\_mode\_t {
  - kFLEXIO\_TimerModeDisabled = 0x0U,
  - kFLEXIO\_TimerModeDual8BitBaudBit = 0x1U,
  - kFLEXIO\_TimerModeDual8BitPWM = 0x2U,
  - kFLEXIO\_TimerModeSingle16Bit = 0x3U }

*Define type of timer work mode.*
- enum flexio\_timer\_output\_t {
  - kFLEXIO\_TimerOutputOneNotAffectedByReset = 0x0U,
  - kFLEXIO\_TimerOutputZeroNotAffectedByReset = 0x1U,
  - kFLEXIO\_TimerOutputOneAffectedByReset = 0x2U,
  - kFLEXIO\_TimerOutputZeroAffectedByReset = 0x3U }

*Define type of timer initial output or timer reset condition.*
- enum flexio\_timer\_decrement\_source\_t {
  - kFLEXIO\_TimerDecSrcOnFlexIOClockShiftTimerOutput = 0x0U,
  - kFLEXIO\_TimerDecSrcOnTriggerInputShiftTimerOutput = 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.*

## Variables

- FLEXIO\_Type \*const s\_flexioBases []  
*Pointers to flexio bases for each instance.*
- const clock\_ip\_name\_t s\_flexioClocks []  
*Pointers to flexio clocks for each instance.*



## Driver version

- #define **FSL\_FLEXIO\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 2))  
*FlexIO driver version 2.0.2.*

## FlexIO Initialization and De-initialization

- void **FLEXIO\_GetDefaultConfig** (**flexio\_config\_t** \*userConfig)  
*Gets the default configuration to configure the FlexIO module.*
- void **FLEXIO\_Init** (**FLEXIO\_Type** \*base, const **flexio\_config\_t** \*userConfig)  
*Configures the FlexIO with a FlexIO configuration.*
- void **FLEXIO\_Deinit** (**FLEXIO\_Type** \*base)  
*Gates the FlexIO clock.*
- **uint32\_t** **FLEXIO\_GetInstance** (**FLEXIO\_Type** \*base)  
*Get instance number for FLEXIO module.*

## FlexIO Basic Operation

- void **FLEXIO\_Reset** (**FLEXIO\_Type** \*base)  
*Resets the FlexIO module.*
- static void **FLEXIO\_Enable** (**FLEXIO\_Type** \*base, bool enable)  
*Enables the FlexIO module operation.*
- void **FLEXIO\_SetShifterConfig** (**FLEXIO\_Type** \*base, **uint8\_t** index, const **flexio\_shifter\_config\_t** \*shifterConfig)  
*Configures the shifter with the shifter configuration.*
- void **FLEXIO\_SetTimerConfig** (**FLEXIO\_Type** \*base, **uint8\_t** index, const **flexio\_timer\_config\_t** \*timerConfig)  
*Configures the timer with the timer configuration.*

## 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 Driver

### FlexIO Status Operation

- static uint32\_t [FLEXIO\\_GetShifterStatusFlags](#) (FLEXIO\_Type \*base)  
*Gets the shifter status flags.*
- static void [FLEXIO\\_ClearShifterStatusFlags](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Clears the shifter status flags.*
- static uint32\_t [FLEXIO\\_GetShifterErrorFlags](#) (FLEXIO\_Type \*base)  
*Gets the shifter error flags.*
- static void [FLEXIO\\_ClearShifterErrorFlags](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Clears the shifter error flags.*
- static uint32\_t [FLEXIO\\_GetTimerStatusFlags](#) (FLEXIO\_Type \*base)  
*Gets the timer status flags.*
- static void [FLEXIO\\_ClearTimerStatusFlags](#) (FLEXIO\_Type \*base, uint32\_t mask)  
*Clears the timer status flags.*

### FlexIO DMA Operation

- static void [FLEXIO\\_EnableShifterStatusDMA](#) (FLEXIO\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the shifter status DMA.*
- uint32\_t [FLEXIO\\_GetShifterBufferAddress](#) (FLEXIO\_Type \*base, [flexio\\_shifter\\_buffer\\_type\\_t](#) type, uint8\_t index)  
*Gets the shifter buffer address for the DMA transfer usage.*
- status\_t [FLEXIO\\_RegisterHandleIRQ](#) (void \*base, void \*handle, [flexio\\_isr\\_t](#) isr)  
*Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.*
- status\_t [FLEXIO\\_UnregisterHandleIRQ](#) (void \*base)  
*Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.*

## 23.2.2 Data Structure Documentation

### 23.2.2.1 struct flexio\_config\_t

#### Data Fields

- bool [enableFlexio](#)  
*Enable/disable FlexIO module.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*
- bool [enableFastAccess](#)  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

### 23.2.2.1.0.37 Field Documentation

#### 23.2.2.1.0.37.1 bool flexio\_config\_t::enableFastAccess

### 23.2.2.2 struct flexio\_timer\_config\_t

#### Data Fields

- uint32\_t [triggerSelect](#)  
*The internal trigger selection number using MACROs.*
- flexio\_timer\_trigger\_polarity\_t [triggerPolarity](#)  
*Trigger Polarity.*
- flexio\_timer\_trigger\_source\_t [triggerSource](#)  
*Trigger Source, internal (see 'trgsel') or external.*
- flexio\_pin\_config\_t [pinConfig](#)  
*Timer Pin Configuration.*
- uint32\_t [pinSelect](#)  
*Timer Pin number Select.*
- flexio\_pin\_polarity\_t [pinPolarity](#)  
*Timer Pin Polarity.*
- flexio\_timer\_mode\_t [timerMode](#)  
*Timer work Mode.*
- flexio\_timer\_output\_t [timerOutput](#)  
*Configures the initial state of the Timer Output and whether it is affected by the Timer reset.*
- flexio\_timer\_decrement\_source\_t [timerDecrement](#)  
*Configures the source of the Timer decrement and the source of the Shift clock.*
- flexio\_timer\_reset\_condition\_t [timerReset](#)  
*Configures the condition that causes the timer counter (and optionally the timer output) to be reset.*
- flexio\_timer\_disable\_condition\_t [timerDisable](#)  
*Configures the condition that causes the Timer to be disabled and stop decrementing.*
- flexio\_timer\_enable\_condition\_t [timerEnable](#)  
*Configures the condition that causes the Timer to be enabled and start decrementing.*
- flexio\_timer\_stop\_bit\_condition\_t [timerStop](#)  
*Timer STOP Bit generation.*
- flexio\_timer\_start\_bit\_condition\_t [timerStart](#)  
*Timer STRAT Bit generation.*
- uint32\_t [timerCompare](#)  
*Value for Timer Compare N Register.*

## FlexIO Driver

### 23.2.2.2.0.38 Field Documentation

- 23.2.2.2.0.38.1 `uint32_t flexio_timer_config_t::triggerSelect`
- 23.2.2.2.0.38.2 `flexio_timer_trigger_polarity_t flexio_timer_config_t::triggerPolarity`
- 23.2.2.2.0.38.3 `flexio_timer_trigger_source_t flexio_timer_config_t::triggerSource`
- 23.2.2.2.0.38.4 `flexio_pin_config_t flexio_timer_config_t::pinConfig`
- 23.2.2.2.0.38.5 `uint32_t flexio_timer_config_t::pinSelect`
- 23.2.2.2.0.38.6 `flexio_pin_polarity_t flexio_timer_config_t::pinPolarity`
- 23.2.2.2.0.38.7 `flexio_timer_mode_t flexio_timer_config_t::timerMode`
- 23.2.2.2.0.38.8 `flexio_timer_output_t flexio_timer_config_t::timerOutput`
- 23.2.2.2.0.38.9 `flexio_timer_decrement_source_t flexio_timer_config_t::timerDecrement`
- 23.2.2.2.0.38.10 `flexio_timer_reset_condition_t flexio_timer_config_t::timerReset`
- 23.2.2.2.0.38.11 `flexio_timer_disable_condition_t flexio_timer_config_t::timerDisable`
- 23.2.2.2.0.38.12 `flexio_timer_enable_condition_t flexio_timer_config_t::timerEnable`
- 23.2.2.2.0.38.13 `flexio_timer_stop_bit_condition_t flexio_timer_config_t::timerStop`
- 23.2.2.2.0.38.14 `flexio_timer_start_bit_condition_t flexio_timer_config_t::timerStart`
- 23.2.2.2.0.38.15 `uint32_t flexio_timer_config_t::timerCompare`

### 23.2.2.3 struct `flexio_shifter_config_t`

#### Data Fields

- `uint32_t timerSelect`  
*Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.*
- `flexio_shifter_timer_polarity_t timerPolarity`  
*Timer Polarity.*
- `flexio_pin_config_t pinConfig`  
*Shifter Pin Configuration.*
- `uint32_t pinSelect`  
*Shifter Pin number Select.*
- `flexio_pin_polarity_t pinPolarity`  
*Shifter Pin Polarity.*
- `flexio_shifter_mode_t shifterMode`  
*Configures the mode of the Shifter.*
- `flexio_shifter_input_source_t inputSource`  
*Selects the input source for the shifter.*

- `flexio_shifter_stop_bit_t` `shifterStop`  
*Shifter STOP bit.*
- `flexio_shifter_start_bit_t` `shifterStart`  
*Shifter START bit.*

### 23.2.2.3.0.39 Field Documentation

23.2.2.3.0.39.1 `uint32_t flexio_shifter_config_t::timerSelect`

23.2.2.3.0.39.2 `flexio_shifter_timer_polarity_t flexio_shifter_config_t::timerPolarity`

23.2.2.3.0.39.3 `flexio_pin_config_t flexio_shifter_config_t::pinConfig`

23.2.2.3.0.39.4 `uint32_t flexio_shifter_config_t::pinSelect`

23.2.2.3.0.39.5 `flexio_pin_polarity_t flexio_shifter_config_t::pinPolarity`

23.2.2.3.0.39.6 `flexio_shifter_mode_t flexio_shifter_config_t::shifterMode`

23.2.2.3.0.39.7 `flexio_shifter_input_source_t flexio_shifter_config_t::inputSource`

23.2.2.3.0.39.8 `flexio_shifter_stop_bit_t flexio_shifter_config_t::shifterStop`

23.2.2.3.0.39.9 `flexio_shifter_start_bit_t flexio_shifter_config_t::shifterStart`

### 23.2.3 Macro Definition Documentation

23.2.3.1 `#define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

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

### 23.2.4 Typedef Documentation

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

### 23.2.5 Enumeration Type Documentation

23.2.5.1 `enum flexio_timer_trigger_polarity_t`

Enumerator

*kFLEXIO\_TimerTriggerPolarityActiveHigh* Active high.

*kFLEXIO\_TimerTriggerPolarityActiveLow* Active low.

## FlexIO Driver

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

Enumerator

*kFLEXIO\_TimerTriggerSourceExternal* External trigger selected.

*kFLEXIO\_TimerTriggerSourceInternal* Internal trigger selected.

### 23.2.5.3 enum flexio\_pin\_config\_t

Enumerator

*kFLEXIO\_PinConfigOutputDisabled* Pin output disabled.

*kFLEXIO\_PinConfigOpenDrainOrBidirection* Pin open drain or bidirectional output enable.

*kFLEXIO\_PinConfigBidirectionOutputData* Pin bidirectional output data.

*kFLEXIO\_PinConfigOutput* Pin output.

### 23.2.5.4 enum flexio\_pin\_polarity\_t

Enumerator

*kFLEXIO\_PinActiveHigh* Active high.

*kFLEXIO\_PinActiveLow* Active low.

### 23.2.5.5 enum flexio\_timer\_mode\_t

Enumerator

*kFLEXIO\_TimerModeDisabled* Timer Disabled.

*kFLEXIO\_TimerModeDual8BitBaudBit* Dual 8-bit counters baud/bit mode.

*kFLEXIO\_TimerModeDual8BitPWM* Dual 8-bit counters PWM mode.

*kFLEXIO\_TimerModeSingle16Bit* Single 16-bit counter mode.

### 23.2.5.6 enum flexio\_timer\_output\_t

Enumerator

*kFLEXIO\_TimerOutputOneNotAffectedByReset* Logic one when enabled and is not affected by timer reset.

*kFLEXIO\_TimerOutputZeroNotAffectedByReset* Logic zero when enabled and is not affected by timer reset.

*kFLEXIO\_TimerOutputOneAffectedByReset* Logic one when enabled and on timer reset.

*kFLEXIO\_TimerOutputZeroAffectedByReset* Logic zero when enabled and on timer reset.

### 23.2.5.7 enum flexio\_timer\_decrement\_source\_t

Enumerator

- kFLEXIO\_TimerDecSrcOnFlexIOClockShiftTimerOutput* Decrement counter on FlexIO clock, Shift clock equals Timer output.
- kFLEXIO\_TimerDecSrcOnTriggerInputShiftTimerOutput* Decrement counter on Trigger input (both edges), Shift clock equals Timer output.
- kFLEXIO\_TimerDecSrcOnPinInputShiftPinInput* Decrement counter on Pin input (both edges), Shift clock equals Pin input.
- kFLEXIO\_TimerDecSrcOnTriggerInputShiftTriggerInput* Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

### 23.2.5.8 enum flexio\_timer\_reset\_condition\_t

Enumerator

- kFLEXIO\_TimerResetNever* Timer never reset.
- kFLEXIO\_TimerResetOnTimerPinEqualToTimerOutput* Timer reset on Timer Pin equal to Timer Output.
- kFLEXIO\_TimerResetOnTimerTriggerEqualToTimerOutput* Timer reset on Timer Trigger equal to Timer Output.
- kFLEXIO\_TimerResetOnTimerPinRisingEdge* Timer reset on Timer Pin rising edge.
- kFLEXIO\_TimerResetOnTimerTriggerRisingEdge* Timer reset on Trigger rising edge.
- kFLEXIO\_TimerResetOnTimerTriggerBothEdge* Timer reset on Trigger rising or falling edge.

### 23.2.5.9 enum flexio\_timer\_disable\_condition\_t

Enumerator

- kFLEXIO\_TimerDisableNever* Timer never disabled.
- kFLEXIO\_TimerDisableOnPreTimerDisable* Timer disabled on Timer N-1 disable.
- kFLEXIO\_TimerDisableOnTimerCompare* Timer disabled on Timer compare.
- kFLEXIO\_TimerDisableOnTimerCompareTriggerLow* Timer disabled on Timer compare and Trigger Low.
- kFLEXIO\_TimerDisableOnPinBothEdge* Timer disabled on Pin rising or falling edge.
- kFLEXIO\_TimerDisableOnPinBothEdgeTriggerHigh* Timer disabled on Pin rising or falling edge provided Trigger is high.
- kFLEXIO\_TimerDisableOnTriggerFallingEdge* Timer disabled on Trigger falling edge.

### 23.2.5.10 enum flexio\_timer\_enable\_condition\_t

Enumerator

- kFLEXIO\_TimerEnabledAlways* Timer always enabled.

## 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.

### 23.2.5.11 enum flexio\_timer\_stop\_bit\_condition\_t

Enumerator

- kFLEXIO\_TimerStopBitDisabled* Stop bit disabled.
- kFLEXIO\_TimerStopBitEnableOnTimerCompare* Stop bit is enabled on timer compare.
- kFLEXIO\_TimerStopBitEnableOnTimerDisable* Stop bit is enabled on timer disable.
- kFLEXIO\_TimerStopBitEnableOnTimerCompareDisable* Stop bit is enabled on timer compare and timer disable.

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

Enumerator

- kFLEXIO\_TimerStartBitDisabled* Start bit disabled.
- kFLEXIO\_TimerStartBitEnabled* Start bit enabled.

### 23.2.5.13 enum flexio\_shifter\_timer\_polarity\_t

### 23.2.5.14 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.

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

Enumerator

- kFLEXIO\_ShifterInputFromPin* Shifter input from pin.
- kFLEXIO\_ShifterInputFromNextShifterOutput* Shifter input from Shifter N+1.



### 23.2.5.16 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.

### 23.2.5.17 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.

### 23.2.5.18 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.

## 23.2.6 Function Documentation

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

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

Example:

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

## FlexIO Driver

### Parameters

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>userConfig</i> | pointer to <a href="#">flexio_config_t</a> structure |
|-------------------|------------------------------------------------------|

### 23.2.6.2 void FLEXIO\_Init ( FLEXIO\_Type \* *base*, const flexio\_config\_t \* *userConfig* )

The configuration structure can be filled by the user or be set with default values by [FLEXIO\\_GetDefaultConfig\(\)](#).

### Example

```
flexio_config_t config = {  
.enableFlexio = true,  
.enableInDoze = false,  
.enableInDebug = true,  
.enableFastAccess = false  
};  
FLEXIO_Configure(base, &config);
```

### Parameters

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>base</i>       | FlexIO peripheral base address                       |
| <i>userConfig</i> | pointer to <a href="#">flexio_config_t</a> structure |

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

Call this API to stop the FlexIO clock.

### Note

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

### Parameters

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

### 23.2.6.4 uint32\_t FLEXIO\_GetInstance ( FLEXIO\_Type \* *base* )

## Parameters

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

**23.2.6.5 void FLEXIO\_Reset ( FLEXIO\_Type \* *base* )**

## Parameters

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

**23.2.6.6 static void FLEXIO\_Enable ( FLEXIO\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

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

**23.2.6.7 void FLEXIO\_SetShifterConfig ( FLEXIO\_Type \* *base*, uint8\_t *index*, const flexio\_shifter\_config\_t \* *shifterConfig* )**

The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

## Example

```
flexio_shifter_config_t config = {
    .timerSelect = 0,
    .timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
    .pinPolarity = kFLEXIO_PinActiveLow,
    .shifterMode = kFLEXIO_ShifterModeTransmit,
    .inputSource = kFLEXIO_ShifterInputFromPin,
    .shifterStop = kFLEXIO_ShifterStopBitHigh,
    .shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

## FlexIO Driver

### Parameters

|                      |                                                              |
|----------------------|--------------------------------------------------------------|
| <i>base</i>          | FlexIO peripheral base address                               |
| <i>index</i>         | Shifter index                                                |
| <i>shifterConfig</i> | Pointer to <a href="#">flexio_shifter_config_t</a> structure |

### 23.2.6.8 void FLEXIO\_SetTimerConfig ( FLEXIO\_Type \* *base*, uint8\_t *index*, const flexio\_timer\_config\_t \* *timerConfig* )

The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

### Example

```
flexio_timer_config_t config = {
    .triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
    .triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
    .triggerSource = kFLEXIO_TimerTriggerSourceInternal,
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
    .pinSelect = 0,
    .pinPolarity = kFLEXIO_PinActiveHigh,
    .timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
    .timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
    .timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
    ,
    .timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
    .timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
    .timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
    .timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
    .timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>base</i>        | FlexIO peripheral base address                                 |
| <i>index</i>       | Timer index                                                    |
| <i>timerConfig</i> | Pointer to the <a href="#">flexio_timer_config_t</a> structure |

### 23.2.6.9 static void FLEXIO\_EnableShifterStatusInterrupts ( FLEXIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt generates when the corresponding SSF is set.

## Parameters

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

## Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.10 `static void FLEXIO_DisableShifterStatusInterrupts ( FLEXIO_Type * base, uint32_t mask ) [inline], [static]`

The interrupt won't generate when the corresponding SSF is set.

## Parameters

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

## Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.11 `static void FLEXIO_EnableShifterErrorInterrupts ( FLEXIO_Type * base, uint32_t mask ) [inline], [static]`

The interrupt generates when the corresponding SEF is set.

## Parameters

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

## Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

## FlexIO Driver

**23.2.6.12** `static void FLEXIO_DisableShifterErrorInterrupts ( FLEXIO_Type * base,  
uint32_t mask ) [inline], [static]`

The interrupt won't generate when the corresponding SEF is set.

## Parameters

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

## Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.13 `static void FLEXIO_EnableTimerStatusInterrupts ( FLEXIO_Type * base, uint32_t mask ) [inline], [static]`

The interrupt generates when the corresponding SSF is set.

## Parameters

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

## Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

### 23.2.6.14 `static void FLEXIO_DisableTimerStatusInterrupts ( FLEXIO_Type * base, uint32_t mask ) [inline], [static]`

The interrupt won't generate when the corresponding SSF is set.

## Parameters

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

## Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

### 23.2.6.15 `static uint32_t FLEXIO_GetShifterStatusFlags ( FLEXIO_Type * base ) [inline], [static]`

## FlexIO Driver

### Parameters

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

### Returns

Shifter status flags

**23.2.6.16** `static void FLEXIO_ClearShifterStatusFlags ( FLEXIO_Type * base, uint32_t mask ) [inline], [static]`

### Parameters

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

### Note

For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

**23.2.6.17** `static uint32_t FLEXIO_GetShifterErrorFlags ( FLEXIO_Type * base ) [inline], [static]`

### Parameters

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

### Returns

Shifter error flags

**23.2.6.18** `static void FLEXIO_ClearShifterErrorFlags ( FLEXIO_Type * base, uint32_t mask ) [inline], [static]`



## Parameters

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

## Note

For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

### 23.2.6.19 `static uint32_t FLEXIO_GetTimerStatusFlags ( FLEXIO_Type * base ) [inline], [static]`

## Parameters

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

## Returns

Timer status flags

### 23.2.6.20 `static void FLEXIO_ClearTimerStatusFlags ( FLEXIO_Type * base, uint32_t mask ) [inline], [static]`

## Parameters

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

## Note

For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using  $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

### 23.2.6.21 `static void FLEXIO_EnableShifterStatusDMA ( FLEXIO_Type * base, uint32_t mask, bool enable ) [inline], [static]`

The DMA request generates when the corresponding SSF is set.

## Note

For multiple shifter status DMA enables, for example, calculate the mask by using  $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

## FlexIO Driver

### Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | FlexIO peripheral base address                                                    |
| <i>mask</i>   | The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$ |
| <i>enable</i> | True to enable, false to disable.                                                 |

### 23.2.6.22 `uint32_t FLEXIO_GetShifterBufferAddress ( FLEXIO_Type * base, flexio_shifter_buffer_type_t type, uint8_t index )`

### Parameters

|              |                                              |
|--------------|----------------------------------------------|
| <i>base</i>  | FlexIO peripheral base address               |
| <i>type</i>  | Shifter type of flexio_shifter_buffer_type_t |
| <i>index</i> | Shifter index                                |

### Returns

Corresponding shifter buffer index

### 23.2.6.23 `status_t FLEXIO_RegisterHandleIRQ ( void * base, void * handle, flexio_isr_t isr )`

### Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | Pointer to the FlexIO simulated peripheral type.        |
| <i>handle</i> | Pointer to the handler for FlexIO simulated peripheral. |
| <i>isr</i>    | FlexIO simulated peripheral interrupt handler.          |

### Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

### 23.2.6.24 `status_t FLEXIO_UnregisterHandleIRQ ( void * base )`

## Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | Pointer to the FlexIO simulated peripheral type. |
|-------------|--------------------------------------------------|

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

## 23.2.7 Variable Documentation

**23.2.7.1 FLEXIO\_Type\* const s\_flexioBases[]**

**23.2.7.2 const clock\_ip\_name\_t s\_flexioClocks[]**

### 23.3 FlexIO Camera Driver

#### 23.3.1 Overview

The MCUXpresso SDK provides a driver for the camera function using Flexible I/O.

FlexIO Camera driver includes functional APIs and eDMA transactional APIs. Functional APIs target low level APIs. Users can use functional APIs for FlexIO Camera initialization/configuration/operation purpose. Using the functional API requires knowledge of the FlexIO Camera peripheral and how to organize functional APIs to meet the requirements of the application. All functional API use the [FLEXIO\\_CAMERA\\_Type](#) \* as the first parameter. FlexIO Camera functional operation groups provide the functional APIs set.

eDMA transactional APIs target high-level APIs. Users can use the transactional API to enable the peripheral quickly and can also use in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_camera_edma_handle_t` as the second parameter. Users need to initialize the handle by calling the [FLEXIO\\_CAMERA\\_TransferCreateHandleEDMA\(\)](#) API.

eDMA transactional APIs support asynchronous receive. This means that the functions [FLEXIO\\_CAMERA\\_TransferReceiveEDMA\(\)](#) set up an interrupt for data receive. When the receive is complete, the upper layer is notified through a callback function with the status `kStatus_FLEXIO_CAMERA_RxIdle`.

#### 23.3.2 Typical use case

##### 23.3.2.1 FlexIO Camera Receive using eDMA method

```
volatile uint32_t isEDMAGetOnePictureFinish = false;
edma_handle_t g_edmaHandle;
flexio_camera_edma_handle_t g_cameraEdmaHandle;
edma_config_t edmaConfig;
FLEXIO_CAMERA_Type g_FlexioCameraDevice = {.flexioBase = FLEXIO0,
                                             .datPinStartIdx = 24U, /* fxio_pin 24 -31 are used. */
                                             .pclkPinIdx = 1U,      /* fxio_pin 1 is used as pclk pin. */
                                             .hrefPinIdx = 18U,     /* flexio_pin 18 is used as href pin. */
                                             .shifterStartIdx = 0U, /* Shifter 0 = 7 are used. */
                                             .shifterCount = 8U,
                                             .timerIdx = 0U};

flexio_camera_config_t cameraConfig;

/* Configure DMAMUX */
DMAMUX_Init(DMAMUX0);
/* Configure DMA */
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(DMA0, &edmaConfig);

DMAMUX_SetSource(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF, (g_FlexioCameraDevice.
    shifterStartIdx + 1U));
DMAMUX_EnableChannel(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);
EDMA_CreateHandle(&g_edmaHandle, DMA0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);

FLEXIO_CAMERA_GetDefaultConfig(&cameraConfig);
FLEXIO_CAMERA_Init(&g_FlexioCameraDevice, &cameraConfig);
/* Clear all the flag. */
```

```

FLEXIO_CAMERA_ClearStatusFlags(&g_FlexioCameraDevice,
                                kFLEXIO_CAMERA_RxDataRegFullFlag |
                                kFLEXIO_CAMERA_RxErrorFlag);
FLEXIO_ClearTimerStatusFlags(FLEXIO0, 0xFF);
FLEXIO_CAMERA_TransferCreateHandleEDMA(&g_FlexioCameraDevice, &
                                        g_cameraEdmaHandle, FLEXIO_CAMERA_UserCallback, NULL,
                                        &g_edmaHandle);
cameraTransfer.dataAddress = (uint32_t)u16CameraFrameBuffer;
cameraTransfer.dataNum = sizeof(u16CameraFrameBuffer);
FLEXIO_CAMERA_TransferReceiveEDMA(&g_FlexioCameraDevice, &
                                   g_cameraEdmaHandle, &cameraTransfer);
while (!(isEDMAGetOnePictureFinish))
{
    ;
}

/* A callback function is also needed */
void FLEXIO_CAMERA_UserCallback(FLEXIO_CAMERA_Type *base,
                                flexio_camera_edma_handle_t *handle,
                                status_t status,
                                void *userData)
{
    userData = userData;
    /* eDMA Transfer finished */
    if (kStatus_FLEXIO_CAMERA_RxIdle == status)
    {
        isEDMAGetOnePictureFinish = true;
    }
}

```

## Modules

- [FlexIO eDMA Camera Driver](#)

## Data Structures

- struct [FLEXIO\\_CAMERA\\_Type](#)  
*Define structure of configuring the FlexIO Camera device. [More...](#)*
- struct [flexio\\_camera\\_config\\_t](#)  
*Define FlexIO Camera user configuration structure. [More...](#)*
- struct [flexio\\_camera\\_transfer\\_t](#)  
*Define FlexIO Camera transfer structure. [More...](#)*

## Macros

- #define [FLEXIO\\_CAMERA\\_PARALLEL\\_DATA\\_WIDTH](#) (8U)  
*Define the Camera CPI interface is constantly 8-bit width.*

## Enumerations

- enum [\\_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, 2))  
*FlexIO Camera driver version 2.1.2.*

## Initialization and configuration

- void `FLEXIO_CAMERA_Init` (`FLEXIO_CAMERA_Type` \*base, const `flexio_camera_config_t` \*config)  
*Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO Camera.*
- void `FLEXIO_CAMERA_Deinit` (`FLEXIO_CAMERA_Type` \*base)  
*Resets the FLEXIO\_CAMERA shifer and timer config.*
- void `FLEXIO_CAMERA_GetDefaultConfig` (`flexio_camera_config_t` \*config)  
*Gets the default configuration to configure the FlexIO Camera.*
- static void `FLEXIO_CAMERA_Enable` (`FLEXIO_CAMERA_Type` \*base, bool enable)  
*Enables/disables the FlexIO Camera module operation.*

## Status

- uint32\_t `FLEXIO_CAMERA_GetStatusFlags` (`FLEXIO_CAMERA_Type` \*base)  
*Gets the FlexIO Camera status flags.*
- void `FLEXIO_CAMERA_ClearStatusFlags` (`FLEXIO_CAMERA_Type` \*base, uint32\_t mask)  
*Clears the receive buffer full flag manually.*

## Interrupts

- void `FLEXIO_CAMERA_EnableInterrupt` (`FLEXIO_CAMERA_Type` \*base)  
*Switches on the interrupt for receive buffer full event.*
- void `FLEXIO_CAMERA_DisableInterrupt` (`FLEXIO_CAMERA_Type` \*base)  
*Switches off the interrupt for receive buffer full event.*

## DMA support

- static void `FLEXIO_CAMERA_EnableRxDMA` (`FLEXIO_CAMERA_Type` \*base, bool enable)

*Enables/disables the FlexIO Camera receive DMA.*

- static uint32\_t **FLEXIO\_CAMERA\_GetRxBufferAddress** (FLEXIO\_CAMERA\_Type \*base)  
*Gets the data from the receive buffer.*

### 23.3.3 Data Structure Documentation

#### 23.3.3.1 struct FLEXIO\_CAMERA\_Type

##### Data Fields

- FLEXIO\_Type \* **flexioBase**  
*FlexIO module base address.*
- uint32\_t **datPinStartIdx**  
*First data pin (D0) index for flexio\_camera.*
- uint32\_t **pclkPinIdx**  
*Pixel clock pin (PCLK) index for flexio\_camera.*
- uint32\_t **hrefPinIdx**  
*Horizontal sync pin (HREF) index for flexio\_camera.*
- uint32\_t **shifterStartIdx**  
*First shifter index used for flexio\_camera data FIFO.*
- uint32\_t **shifterCount**  
*The count of shifters that are used as flexio\_camera data FIFO.*
- uint32\_t **timerIdx**  
*Timer index used for flexio\_camera in FlexIO.*

##### 23.3.3.1.0.40 Field Documentation

###### 23.3.3.1.0.40.1 FLEXIO\_Type\* FLEXIO\_CAMERA\_Type::flexioBase

###### 23.3.3.1.0.40.2 uint32\_t FLEXIO\_CAMERA\_Type::datPinStartIdx

Then the successive following FLEXIO\_CAMERA\_DATA\_WIDTH-1 pins are used as D1-D7.

###### 23.3.3.1.0.40.3 uint32\_t FLEXIO\_CAMERA\_Type::pclkPinIdx

###### 23.3.3.1.0.40.4 uint32\_t FLEXIO\_CAMERA\_Type::hrefPinIdx

###### 23.3.3.1.0.40.5 uint32\_t FLEXIO\_CAMERA\_Type::shifterStartIdx

###### 23.3.3.1.0.40.6 uint32\_t FLEXIO\_CAMERA\_Type::shifterCount

###### 23.3.3.1.0.40.7 uint32\_t FLEXIO\_CAMERA\_Type::timerIdx

#### 23.3.3.2 struct flexio\_camera\_config\_t

##### Data Fields

- bool **enablecamera**  
*Enable/disable FlexIO Camera TX & RX.*

## 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.*

### 23.3.3.2.0.41 Field Documentation

23.3.3.2.0.41.1 bool [flexio\\_camera\\_config\\_t::enablecamera](#)

23.3.3.2.0.41.2 bool [flexio\\_camera\\_config\\_t::enableFastAccess](#)

### 23.3.3.3 struct [flexio\\_camera\\_transfer\\_t](#)

#### Data Fields

- uint32\_t [dataAddress](#)  
*Transfer buffer.*
- uint32\_t [dataNum](#)  
*Transfer num.*

### 23.3.4 Macro Definition Documentation

23.3.4.1 #define [FSL\\_FLEXIO\\_CAMERA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 1, 2))

23.3.4.2 #define [FLEXIO\\_CAMERA\\_PARALLEL\\_DATA\\_WIDTH](#) (8U)

### 23.3.5 Enumeration Type Documentation

#### 23.3.5.1 enum [\\_flexio\\_camera\\_status](#)

Enumerator

*kStatus\_FLEXIO\_CAMERA\_RxBusy* Receiver is busy.  
*kStatus\_FLEXIO\_CAMERA\_RxIdle* Camera receiver is idle.

#### 23.3.5.2 enum [\\_flexio\\_camera\\_status\\_flags](#)

Enumerator

*kFLEXIO\_CAMERA\_RxDataRegFullFlag* Receive buffer full flag.  
*kFLEXIO\_CAMERA\_RxErrorFlag* Receive buffer error flag.



## 23.3.6 Function Documentation

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

## 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 |

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

Note

After calling this API, call [FLEXIO\\_CAMERA\\_Init](#) to use the FlexIO Camera module.

Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
|-------------|---------------------------------------------------------|

### 23.3.6.3 void FLEXIO\_CAMERA\_GetDefaultConfig ( flexio\_camera\_config\_t \* *config* )

The configuration can be used directly for calling the [FLEXIO\\_CAMERA\\_Init\(\)](#). Example:

```
flexio_camera_config_t config;  
FLEXIO_CAMERA_GetDefaultConfig(&userConfig);
```

Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>config</i> | Pointer to the <a href="#">flexio_camera_config_t</a> structure |
|---------------|-----------------------------------------------------------------|

### 23.3.6.4 static void FLEXIO\_CAMERA\_Enable ( FLEXIO\_CAMERA\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> |
| <i>enable</i> | True to enable, false does not have any effect.   |

### 23.3.6.5 uint32\_t FLEXIO\_CAMERA\_GetStatusFlags ( FLEXIO\_CAMERA\_Type \* *base* )

## Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_CAMERA_Type</a> structure |
|-------------|---------------------------------------------------------|

## Returns

FlexIO shifter status flags

- FLEXIO\_SHIFTSTAT\_SSF\_MASK
- 0

**23.3.6.6 void FLEXIO\_CAMERA\_ClearStatusFlags ( FLEXIO\_CAMERA\_Type \* *base*, uint32\_t *mask* )**

## Parameters

|             |                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the device.                                                                                                                                                                               |
| <i>mask</i> | status flag The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_CAMERA_RxDataRegFullFlag</li> <li>• kFLEXIO_CAMERA_RxErrorFlag</li> </ul> |

**23.3.6.7 void FLEXIO\_CAMERA\_EnableInterrupt ( FLEXIO\_CAMERA\_Type \* *base* )**

## Parameters

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

**23.3.6.8 void FLEXIO\_CAMERA\_DisableInterrupt ( FLEXIO\_CAMERA\_Type \* *base* )**

## Parameters

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

**23.3.6.9 static void FLEXIO\_CAMERA\_EnableRxDMA ( FLEXIO\_CAMERA\_Type \* *base*, bool *enable* ) [inline], [static]**

## 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 channels, one for transferring data from Camera, such as 0V7670 to FlexIO buffer, another is for transferring data from FlexIO buffer to LCD.

### 23.3.6.10 `static uint32_t FLEXIO_CAMERA_GetRxBufferAddress ( FLEXIO_CAMERA_Type * base ) [inline], [static]`

### Parameters

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

### Returns

data Pointer to the buffer that keeps the data with count of `base->shifterCount` .

## 23.3.7 FlexIO eDMA Camera Driver

### 23.3.7.1 Overview

#### Data Structures

- struct `flexio_camera_edma_handle_t`  
*Camera eDMA handle. [More...](#)*

#### Typedefs

- typedef void(`* flexio_camera_edma_transfer_callback_t`)(`FLEXIO_CAMERA_Type *base`, `flexio_camera_edma_handle_t *handle`, `status_t status`, `void *userData`)  
*Camera transfer callback function.*

#### Driver version

- #define `FSL_FLEXIO_CAMERA_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 2))  
*FlexIO Camera EDMA driver version 2.1.2.*

#### eDMA transactional

- status\_t `FLEXIO_CAMERA_TransferCreateHandleEDMA` (`FLEXIO_CAMERA_Type *base`, `flexio_camera_edma_handle_t *handle`, `flexio_camera_edma_transfer_callback_t callback`, `void *userData`, `edma_handle_t *rxEdmaHandle`)  
*Initializes the Camera handle, which is used in transactional functions.*
- status\_t `FLEXIO_CAMERA_TransferReceiveEDMA` (`FLEXIO_CAMERA_Type *base`, `flexio_camera_edma_handle_t *handle`, `flexio_camera_transfer_t *xfer`)  
*Receives data using eDMA.*
- void `FLEXIO_CAMERA_TransferAbortReceiveEDMA` (`FLEXIO_CAMERA_Type *base`, `flexio_camera_edma_handle_t *handle`)  
*Aborts the receive data which used the eDMA.*
- status\_t `FLEXIO_CAMERA_TransferGetReceiveCountEDMA` (`FLEXIO_CAMERA_Type *base`, `flexio_camera_edma_handle_t *handle`, `size_t *count`)  
*Gets the remaining bytes to be received.*

### 23.3.7.2 Data Structure Documentation

#### 23.3.7.2.1 struct `flexio_camera_edma_handle`

Forward declaration of the handle typedef.

#### Data Fields

- `flexio_camera_edma_transfer_callback_t callback`

## FlexIO Camera Driver

- *Callback function.*  
void \* [userData](#)
- *Camera callback function parameter.*  
size\_t [rxSize](#)  
*Total bytes to be received.*
- [edma\\_handle\\_t](#) \* [rxEdmaHandle](#)  
*The eDMA RX channel used.*
- uint8\_t [nbytes](#)  
*eDMA minor byte transfer count initially configured.*
- volatile uint8\_t [rxState](#)  
*RX transfer state.*

### 23.3.7.2.1.1 Field Documentation

23.3.7.2.1.1.1 [flexio\\_camera\\_edma\\_transfer\\_callback\\_t](#) [flexio\\_camera\\_edma\\_handle\\_t::callback](#)

23.3.7.2.1.1.2 void\* [flexio\\_camera\\_edma\\_handle\\_t::userData](#)

23.3.7.2.1.1.3 size\_t [flexio\\_camera\\_edma\\_handle\\_t::rxSize](#)

23.3.7.2.1.1.4 [edma\\_handle\\_t](#)\* [flexio\\_camera\\_edma\\_handle\\_t::rxEdmaHandle](#)

23.3.7.2.1.1.5 uint8\_t [flexio\\_camera\\_edma\\_handle\\_t::nbytes](#)

### 23.3.7.3 Macro Definition Documentation

23.3.7.3.1 #define [FSL\\_FLEXIO\\_CAMERA\\_EDMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 2))

### 23.3.7.4 Typedef Documentation

23.3.7.4.1 typedef void(\* [flexio\\_camera\\_edma\\_transfer\\_callback\\_t](#))([FLEXIO\\_CAMERA\\_Type](#) \*[base](#), [flexio\\_camera\\_edma\\_handle\\_t](#) \*[handle](#), [status\\_t](#) [status](#), void \*[userData](#))

### 23.3.7.5 Function Documentation

23.3.7.5.1 [status\\_t](#) [FLEXIO\\_CAMERA\\_TransferCreateHandleEDMA](#) ( [FLEXIO\\_CAMERA\\_Type](#) \* [base](#), [flexio\\_camera\\_edma\\_handle\\_t](#) \* [handle](#), [flexio\\_camera\\_edma\\_transfer\\_callback\\_t](#) [callback](#), void \* [userData](#), [edma\\_handle\\_t](#) \* [rxEdmaHandle](#) )

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> . |
|-------------|-----------------------------------------------------|

|                     |                                                   |
|---------------------|---------------------------------------------------|
| <i>handle</i>       | Pointer to flexio_camera_edma_handle_t structure. |
| <i>callback</i>     | The callback function.                            |
| <i>userData</i>     | The parameter of the callback function.           |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.    |

Return values

|                           |                                                        |
|---------------------------|--------------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                        |
| <i>kStatus_OutOfRange</i> | The FlexIO Camera eDMA type/handle table out of range. |

### 23.3.7.5.2 **status\_t FLEXIO\_CAMERA\_TransferReceiveEDMA ( FLEXIO\_CAMERA\_Type \* *base*, flexio\_camera\_edma\_handle\_t \* *handle*, flexio\_camera\_transfer\_t \* *xfer* )**

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> .                            |
| <i>handle</i> | Pointer to the flexio_camera_edma_handle_t structure.                          |
| <i>xfer</i>   | Camera eDMA transfer structure, see <a href="#">flexio_camera_transfer_t</a> . |

Return values

|                               |                              |
|-------------------------------|------------------------------|
| <i>kStatus_Success</i>        | if succeeded, others failed. |
| <i>kStatus_CAMERA_Rx-Busy</i> | Previous transfer on going.  |

### 23.3.7.5.3 **void FLEXIO\_CAMERA\_TransferAbortReceiveEDMA ( FLEXIO\_CAMERA\_Type \* *base*, flexio\_camera\_edma\_handle\_t \* *handle* )**

This function aborts the receive data which used the eDMA.

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> . |
|-------------|-----------------------------------------------------|

## FlexIO Camera Driver

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>handle</i> | Pointer to the flexio_camera_edma_handle_t structure. |
|---------------|-------------------------------------------------------|

**23.3.7.5.4 status\_t FLEXIO\_CAMERA\_TransferGetReceiveCountEDMA ( FLEXIO\_CAMERA\_Type \* *base*, flexio\_camera\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the number of bytes still not received.

Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_CAMERA_Type</a> .          |
| <i>handle</i> | Pointer to the flexio_camera_edma_handle_t structure.        |
| <i>count</i>  | Number of bytes sent so far by the non-blocking transaction. |

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Succeed get the transfer count. |
| <i>kStatus_InvalidArgument</i> | The count parameter is invalid. |



## 23.4 FlexIO I2C Master Driver

### 23.4.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2C master function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO I2C master driver includes functional APIs and transactional APIs.

Functional APIs target low level APIs. Functional APIs can be used for the FlexIO I2C master initialization/configuration/operation for the optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO I2C master peripheral and how to organize functional APIs to meet the application requirements. The FlexIO I2C master functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support an asynchronous transfer. This means that the functions `FLEXIO_I2C_MasterTransferNonBlocking()` set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_Success` status.

### 23.4.2 Typical use case

#### 23.4.2.1 FlexIO I2C master transfer using an interrupt method

```
flexio_i2c_master_handle_t g_m_handle;
flexio_i2c_master_config_t masterConfig;
flexio_i2c_master_transfer_t masterXfer;
volatile bool completionFlag = false;
const uint8_t sendData[] = {.....};
FLEXIO_I2C_Type i2cDev;

void FLEXIO_I2C_MasterCallback(FLEXIO_I2C_Type *base, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_Success == status)
    {
        completionFlag = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2C_MasterGetDefaultConfig(&masterConfig);

    FLEXIO_I2C_MasterInit(&i2cDev, &user_config);
    FLEXIO_I2C_MasterTransferCreateHandle(&i2cDev, &g_m_handle,
        FLEXIO_I2C_MasterCallback, NULL);

    // Prepares to send.
```

## 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, 5))
 

*FlexIO I2C master driver version 2.1.5.*

## Initialization and deinitialization

- status\_t FLEXIO\_I2C\_MasterInit (FLEXIO\_I2C\_Type \*base, flexio\_i2c\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)
 

*Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.*
- void FLEXIO\_I2C\_MasterDeinit (FLEXIO\_I2C\_Type \*base)
 

*De-initializes the FlexIO I2C master peripheral.*
- void FLEXIO\_I2C\_MasterGetDefaultConfig (flexio\_i2c\_master\_config\_t \*masterConfig)
 

*Gets the default configuration to configure the FlexIO module.*
- static void FLEXIO\_I2C\_MasterEnable (FLEXIO\_I2C\_Type \*base, bool enable)
 

*Enables/disables the FlexIO module operation.*

## Status

- uint32\_t FLEXIO\_I2C\_MasterGetStatusFlags (FLEXIO\_I2C\_Type \*base)
 

*Gets the FlexIO I2C master status flags.*
- void FLEXIO\_I2C\_MasterClearStatusFlags (FLEXIO\_I2C\_Type \*base, uint32\_t mask)
 

*Clears the FlexIO I2C master status flags.*

## Interrupts

- void FLEXIO\_I2C\_MasterEnableInterrupts (FLEXIO\_I2C\_Type \*base, uint32\_t mask)
 

*Enables the FlexIO i2c master interrupt requests.*
- void FLEXIO\_I2C\_MasterDisableInterrupts (FLEXIO\_I2C\_Type \*base, uint32\_t mask)
 

*Disables the FlexIO I2C master interrupt requests.*

### Bus Operations

- void `FLEXIO_I2C_MasterSetBaudRate` (`FLEXIO_I2C_Type *base`, `uint32_t baudRate_Bps`, `uint32_t srcClock_Hz`)  
*Sets the FlexIO I2C master transfer baudrate.*
- void `FLEXIO_I2C_MasterStart` (`FLEXIO_I2C_Type *base`, `uint8_t address`, `flexio_i2c_direction_t direction`)  
*Sends START + 7-bit address to the bus.*
- void `FLEXIO_I2C_MasterStop` (`FLEXIO_I2C_Type *base`)  
*Sends the stop signal on the bus.*
- void `FLEXIO_I2C_MasterRepeatedStart` (`FLEXIO_I2C_Type *base`)  
*Sends the repeated start signal on the bus.*
- void `FLEXIO_I2C_MasterAbortStop` (`FLEXIO_I2C_Type *base`)  
*Sends the stop signal when transfer is still on-going.*
- void `FLEXIO_I2C_MasterEnableAck` (`FLEXIO_I2C_Type *base`, `bool enable`)  
*Configures the sent ACK/NAK for the following byte.*
- `status_t FLEXIO_I2C_MasterSetTransferCount` (`FLEXIO_I2C_Type *base`, `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.*

### 23.4.3 Data Structure Documentation

#### 23.4.3.1 struct FLEXIO\_I2C\_Type

##### Data Fields

- FLEXIO\_Type \* [flexioBase](#)  
*FlexIO base pointer.*
- uint8\_t [SDAPinIndex](#)  
*Pin select for I2C SDA.*
- uint8\_t [SCLPinIndex](#)  
*Pin select for I2C SCL.*
- uint8\_t [shifterIndex](#) [2]  
*Shifter index used in FlexIO I2C.*
- uint8\_t [timerIndex](#) [2]  
*Timer index used in FlexIO I2C.*

##### 23.4.3.1.0.1 Field Documentation

23.4.3.1.0.1.1 FLEXIO\_Type\* FLEXIO\_I2C\_Type::flexioBase

23.4.3.1.0.1.2 uint8\_t FLEXIO\_I2C\_Type::SDAPinIndex

23.4.3.1.0.1.3 uint8\_t FLEXIO\_I2C\_Type::SCLPinIndex

23.4.3.1.0.1.4 uint8\_t FLEXIO\_I2C\_Type::shifterIndex[2]

23.4.3.1.0.1.5 uint8\_t FLEXIO\_I2C\_Type::timerIndex[2]

#### 23.4.3.2 struct flexio\_i2c\_master\_config\_t

##### Data Fields

- bool [enableMaster](#)  
*Enables the FlexIO I2C peripheral at initialization time.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*
- bool [enableFastAccess](#)  
*Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud rate in Bps.*

## FlexIO I2C Master Driver

### 23.4.3.2.0.2 Field Documentation

**23.4.3.2.0.2.1** `bool flexio_i2c_master_config_t::enableMaster`

**23.4.3.2.0.2.2** `bool flexio_i2c_master_config_t::enableInDoze`

**23.4.3.2.0.2.3** `bool flexio_i2c_master_config_t::enableInDebug`

**23.4.3.2.0.2.4** `bool flexio_i2c_master_config_t::enableFastAccess`

**23.4.3.2.0.2.5** `uint32_t flexio_i2c_master_config_t::baudRate_Bps`

### 23.4.3.3 struct flexio\_i2c\_master\_transfer\_t

#### Data Fields

- `uint32_t flags`  
*Transfer flag which controls the transfer, reserved for FlexIO I2C.*
- `uint8_t slaveAddress`  
*7-bit slave address.*
- `flexio_i2c_direction_t direction`  
*Transfer direction, read or write.*
- `uint32_t subaddress`  
*Sub address.*
- `uint8_t subaddressSize`  
*Size of command buffer.*
- `uint8_t volatile * data`  
*Transfer buffer.*
- `volatile size_t dataSize`  
*Transfer size.*

### 23.4.3.3.0.3 Field Documentation

**23.4.3.3.0.3.1** `uint32_t flexio_i2c_master_transfer_t::flags`

**23.4.3.3.0.3.2** `uint8_t flexio_i2c_master_transfer_t::slaveAddress`

**23.4.3.3.0.3.3** `flexio_i2c_direction_t flexio_i2c_master_transfer_t::direction`

**23.4.3.3.0.3.4** `uint32_t flexio_i2c_master_transfer_t::subaddress`

Transferred MSB first.

**23.4.3.3.0.3.5** `uint8_t flexio_i2c_master_transfer_t::subaddressSize`

**23.4.3.3.0.3.6** `uint8_t volatile* flexio_i2c_master_transfer_t::data`

**23.4.3.3.0.3.7** `volatile size_t flexio_i2c_master_transfer_t::dataSize`

#### **23.4.3.4** `struct flexio_i2c_master_handle`

FlexIO I2C master handle typedef.

#### **Data Fields**

- [flexio\\_i2c\\_master\\_transfer\\_t transfer](#)  
*FlexIO I2C master transfer copy.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t state`  
*Transfer state maintained during transfer.*
- [flexio\\_i2c\\_master\\_transfer\\_callback\\_t completionCallback](#)  
*Callback function called at transfer event.*
- `void * userData`  
*Callback parameter passed to callback function.*

#### **23.4.3.4.0.4** **Field Documentation**

**23.4.3.4.0.4.1** `flexio_i2c_master_transfer_t flexio_i2c_master_handle_t::transfer`

**23.4.3.4.0.4.2** `size_t flexio_i2c_master_handle_t::transferSize`

**23.4.3.4.0.4.3** `uint8_t flexio_i2c_master_handle_t::state`

**23.4.3.4.0.4.4** `flexio_i2c_master_transfer_callback_t flexio_i2c_master_handle_t::completion-  
Callback`

Callback function called at transfer event.

## FlexIO I2C Master Driver

23.4.3.4.0.4.5 void\* flexio\_i2c\_master\_handle\_t::userData

### 23.4.4 Macro Definition Documentation

23.4.4.1 #define FSL\_FLEXIO\_I2C\_MASTER\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 5))

### 23.4.5 Typedef Documentation

23.4.5.1 typedef void(\* flexio\_i2c\_master\_transfer\_callback\_t)(FLEXIO\_I2C\_Type \*base, flexio\_i2c\_master\_handle\_t \*handle, status\_t status, void \*userData)

### 23.4.6 Enumeration Type Documentation

#### 23.4.6.1 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.

#### 23.4.6.2 enum \_flexio\_i2c\_master\_interrupt

Enumerator

*kFLEXIO\_I2C\_TxEmptyInterruptEnable* Tx buffer empty interrupt enable.

*kFLEXIO\_I2C\_RxFullInterruptEnable* Rx buffer full interrupt enable.

#### 23.4.6.3 enum \_flexio\_i2c\_master\_status\_flags

Enumerator

*kFLEXIO\_I2C\_TxEmptyFlag* Tx shifter empty flag.

*kFLEXIO\_I2C\_RxFullFlag* Rx shifter full/Transfer complete flag.

*kFLEXIO\_I2C\_ReceiveNakFlag* Receive NAK flag.

#### 23.4.6.4 enum flexio\_i2c\_direction\_t

Enumerator

*kFLEXIO\_I2C\_Write* Master send to slave.

*kFLEXIO\_I2C\_Read* Master receive from slave.



## 23.4.7 Function Documentation

### 23.4.7.1 `status_t FLEXIO_I2C_MasterInit ( FLEXIO_I2C_Type * base, flexio_i2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

Example

```
FLEXIO_I2C_Type base = {
    .flexioBase = FLEXIO,
    .SDAPinIndex = 0,
    .SCLPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <i>base</i>         | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.            |
| <i>masterConfig</i> | Pointer to <a href="#">flexio_i2c_master_config_t</a> structure. |
| <i>srcClock_Hz</i>  | FlexIO source clock in Hz.                                       |

Return values

|                                |                                                |
|--------------------------------|------------------------------------------------|
| <i>kStatus_Success</i>         | Initialization successful                      |
| <i>kStatus_InvalidArgument</i> | The source clock exceed upper range limitation |

### 23.4.7.2 `void FLEXIO_I2C_MasterDeinit ( FLEXIO_I2C_Type * base )`

Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the [FLEXIO\\_I2C\\_MasterInit](#) is called.

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### 23.4.7.3 `void FLEXIO_I2C_MasterGetDefaultConfig ( flexio_i2c_master_config_t * masterConfig )`

The configuration can be used directly for calling the [FLEXIO\\_I2C\\_MasterInit\(\)](#).

## FlexIO I2C Master Driver

Example:

```
flexio_i2c_master_config_t config;  
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <i>masterConfig</i> | Pointer to <a href="#">flexio_i2c_master_config_t</a> structure. |
|---------------------|------------------------------------------------------------------|

**23.4.7.4 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 does not have any effect. |

**23.4.7.5 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](#) can get the related status.

**23.4.7.6 void FLEXIO\_I2C\_MasterClearStatusFlags ( FLEXIO\_I2C\_Type \* *base*, uint32\_t *mask* )**

Parameters

|             |                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_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_I2C_RxFullFlag</li><li>• kFLEXIO_I2C_ReceiveNakFlag</li></ul> |

**23.4.7.7 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> |

#### 23.4.7.8 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.                                     |

#### 23.4.7.9 void FLEXIO\_I2C\_MasterSetBaudRate ( FLEXIO\_I2C\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

## Parameters

|                     |                                                      |
|---------------------|------------------------------------------------------|
| <i>base</i>         | Pointer to <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                                   |

#### 23.4.7.10 void FLEXIO\_I2C\_MasterStart ( FLEXIO\_I2C\_Type \* *base*, uint8\_t *address*, flexio\_i2c\_direction\_t *direction* )

## Note

This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO\_I2C\_-RxFullFlag status is asserted before calling this API.

## 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> |

### 23.4.7.11 void FLEXIO\_I2C\_MasterStop ( FLEXIO\_I2C\_Type \* *base* )

#### Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### 23.4.7.12 void FLEXIO\_I2C\_MasterRepeatedStart ( FLEXIO\_I2C\_Type \* *base* )

#### Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### 23.4.7.13 void FLEXIO\_I2C\_MasterAbortStop ( FLEXIO\_I2C\_Type \* *base* )

#### Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### 23.4.7.14 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. |

### 23.4.7.15 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.         |

**23.4.7.16** `static void FLEXIO_I2C_MasterWriteByte ( FLEXIO_I2C_Type * base, uint32_t data ) [inline], [static]`

## Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>data</i> | a byte of data.                                       |

**23.4.7.17** `static uint8_t FLEXIO_I2C_MasterReadByte ( FLEXIO_I2C_Type * base ) [inline], [static]`

## Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

## FlexIO I2C Master Driver

### Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

### Returns

data byte read.

### 23.4.7.18 **status\_t FLEXIO\_I2C\_MasterWriteBlocking ( FLEXIO\_I2C\_Type \* *base*, const uint8\_t \* *txBuff*, uint8\_t *txSize* )**

#### Note

This function blocks via polling until all bytes have been sent.

### Parameters

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

### 23.4.7.19 **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.              |

#### 23.4.7.20 **status\_t FLEXIO\_I2C\_MasterTransferBlocking ( FLEXIO\_I2C\_Type \* *base*, flexio\_i2c\_master\_transfer\_t \* *xfer* )**

Note

The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | pointer to <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](#).

#### 23.4.7.21 **status\_t FLEXIO\_I2C\_MasterTransferCreateHandle ( FLEXIO\_I2C\_Type \* *base*, flexio\_i2c\_master\_handle\_t \* *handle*, flexio\_i2c\_master\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

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

#### 23.4.7.22 **status\_t FLEXIO\_I2C\_MasterTransferNonBlocking ( FLEXIO\_I2C\_Type \* *base*, flexio\_i2c\_master\_handle\_t \* *handle*, flexio\_i2c\_master\_transfer\_t \* *xfer* )**

## FlexIO I2C Master Driver

### 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.

### Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure                                         |
| <i>handle</i> | Pointer to <code>flexio_i2c_master_handle_t</code> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <code>flexio_i2c_master_transfer_t</code> structure                               |

### Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_FLEXIO_I2C_Busy</i> | FlexIO I2C is not idle, is running another transfer. |

### 23.4.7.23 `status_t FLEXIO_I2C_MasterTransferGetCount ( FLEXIO_I2C_Type * base, flexio_i2c_master_handle_t * handle, size_t * count )`

### Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                         |
| <i>handle</i> | Pointer to <code>flexio_i2c_master_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                           |

### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

### 23.4.7.24 `void FLEXIO_I2C_MasterTransferAbort ( FLEXIO_I2C_Type * base, flexio_i2c_master_handle_t * handle )`

### Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.



Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure                                         |
| <i>handle</i> | Pointer to <code>flexio_i2c_master_handle_t</code> structure which stores the transfer state |

**23.4.7.25** `void FLEXIO_I2C_MasterTransferHandleIRQ ( void * i2cType, void * i2cHandle )`

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 |

## FlexIO I2S Driver

### 23.5 FlexIO I2S Driver

#### 23.5.1 Overview

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

The FlexIO I2S driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs.

Functional APIs can be used for FlexIO I2S initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO I2S peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO I2S functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `sai_handle_t` as the first parameter. Initialize the handle by calling the `FlexIO_I2S_TransferTxCreateHandle()` or `FlexIO_I2S_TransferRxCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO\\_I2S\\_TransferSendNonBlocking\(\)](#) and [FLEXIO\\_I2S\\_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXIO_I2S_TxIdle` and `kStatus_FLEXIO_I2S_RxIdle` status.

#### 23.5.2 Typical use case

##### 23.5.2.1 FlexIO I2S send/receive using an interrupt method

```
sai_handle_t g_saiTxHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
volatile bool rxFinished;
const uint8_t sendData[] = [.....];

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);
```

```

FLEXIO_I2S_TxInit(FLEXIO_I2S0, &user_config);
FLEXIO_I2S_TransferTxCreateHandle(FLEXIO_I2S0, &g_saiHandle,
    FLEXIO_I2S_UserCallback, NULL);

//Configures the SAI format.
FLEXIO_I2S_TransferTxSetTransferFormat(FLEXIO_I2S0, &g_saiHandle, mclkSource, mclk);

// Prepares to send.
sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S0, &g_saiHandle, &
    sendXfer);

// Waiting to send is finished.
while (!txFinished)
{
}

// ...
}

```

### 23.5.2.2 FLEXIO\_I2S send/receive using a DMA method

```

sai_handle_t g_saiHandle;
dma_handle_t g_saiTxDmaHandle;
dma_handle_t g_saiRxDmaHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
uint8_t sendData[] = ...;

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_I2S_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2S_TxGetDefaultConfig(&user_config);
    FLEXIO_I2S_TxInit(FLEXIO_I2S0, &user_config);

    // Sets up the DMA.
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL, FLEXIO_I2S_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL);

    DMA_Init(DMA0);

    /* Creates the DMA handle. */
    DMA_TransferTxCreateHandle(&g_saiTxDmaHandle, DMA0, FLEXIO_I2S_TX_DMA_CHANNEL);

    FLEXIO_I2S_TransferTxCreateHandleDMA(FLEXIO_I2S0, &g_saiTxDmaHandle
        , FLEXIO_I2S_UserCallback, NULL);
}

```

## FlexIO I2S Driver

```
// 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, 4))  
*FlexIO I2S driver version 2.1.4.*

### 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 data using a blocking method.*
- static void [FLEXIO\\_I2S\\_WriteData](#) ([FLEXIO\\_I2S\\_Type](#) \*base, [uint8\\_t](#) bitWidth, [uint32\\_t](#) data)

- *Writes data into a data register.*
- 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.*

## 23.5.3 Data Structure Documentation

### 23.5.3.1 struct FLEXIO\_I2S\_Type

#### Data Fields

- `FLEXIO_Type * flexioBase`
  - *FlexIO base pointer.*

## FlexIO I2S Driver

- uint8\_t [txPinIndex](#)  
*Tx data pin index in FlexIO pins.*
- uint8\_t [rxPinIndex](#)  
*Rx data pin index.*
- uint8\_t [bclkPinIndex](#)  
*Bit clock pin index.*
- uint8\_t [fsPinIndex](#)  
*Frame sync pin index.*
- uint8\_t [txShifterIndex](#)  
*Tx data shifter index.*
- uint8\_t [rxShifterIndex](#)  
*Rx data shifter index.*
- uint8\_t [bclkTimerIndex](#)  
*Bit clock timer index.*
- uint8\_t [fsTimerIndex](#)  
*Frame sync timer index.*

### 23.5.3.2 struct flexio\_i2s\_config\_t

#### Data Fields

- bool [enableI2S](#)  
*Enable FlexIO I2S.*
- [flexio\\_i2s\\_master\\_slave\\_t](#) masterSlave  
*Master or slave.*
- [flexio\\_pin\\_polarity\\_t](#) txPinPolarity  
*Tx data pin polarity, active high or low.*
- [flexio\\_pin\\_polarity\\_t](#) rxPinPolarity  
*Rx data pin polarity.*
- [flexio\\_pin\\_polarity\\_t](#) bclkPinPolarity  
*Bit clock pin polarity.*
- [flexio\\_pin\\_polarity\\_t](#) fsPinPolarity  
*Frame sync pin polarity.*
- [flexio\\_shifter\\_timer\\_polarity\\_t](#) txTimerPolarity  
*Tx data valid on bclk rising or falling edge.*
- [flexio\\_shifter\\_timer\\_polarity\\_t](#) rxTimerPolarity  
*Rx data valid on bclk rising or falling edge.*

### 23.5.3.3 struct flexio\_i2s\_format\_t

#### Data Fields

- uint8\_t [bitWidth](#)  
*Bit width of audio data, always 8/16/24/32 bits.*
- uint32\_t [sampleRate\\_Hz](#)  
*Sample rate of the audio data.*



### 23.5.3.4 struct flexio\_i2s\_transfer\_t

#### Data Fields

- uint8\_t \* [data](#)  
*Data buffer start pointer.*
- size\_t [dataSize](#)  
*Bytes to be transferred.*

#### 23.5.3.4.0.5 Field Documentation

##### 23.5.3.4.0.5.1 size\_t flexio\_i2s\_transfer\_t::dataSize

### 23.5.3.5 struct \_flexio\_i2s\_handle

#### Data Fields

- uint32\_t [state](#)  
*Internal state.*
- [flexio\\_i2s\\_callback\\_t](#) [callback](#)  
*Callback function called at transfer event.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*
- uint8\_t [bitWidth](#)  
*Bit width for transfer, 8/16/24/32bits.*
- [flexio\\_i2s\\_transfer\\_t](#) [queue](#) [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*

## 23.5.4 Macro Definition Documentation

### 23.5.4.1 #define FSL\_FLEXIO\_I2S\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 4))

### 23.5.4.2 #define FLEXIO\_I2S\_XFER\_QUEUE\_SIZE (4)

## 23.5.5 Enumeration Type Documentation

### 23.5.5.1 enum \_flexio\_i2s\_status

Enumerator

- kStatus\_FLEXIO\_I2S\_Idle* FlexIO I2S is in idle state.
- kStatus\_FLEXIO\_I2S\_TxBusy* FlexIO I2S Tx is busy.

## FlexIO I2S Driver

*kStatus\_FLEXIO\_I2S\_RxBusy* FlexIO I2S Tx is busy.  
*kStatus\_FLEXIO\_I2S\_Error* FlexIO I2S error occurred.  
*kStatus\_FLEXIO\_I2S\_QueueFull* FlexIO I2S transfer queue is full.

### 23.5.5.2 enum flexio\_i2s\_master\_slave\_t

Enumerator

*kFLEXIO\_I2S\_Master* Master mode.  
*kFLEXIO\_I2S\_Slave* Slave mode.

### 23.5.5.3 enum \_flexio\_i2s\_interrupt\_enable

Enumerator

*kFLEXIO\_I2S\_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.  
*kFLEXIO\_I2S\_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

### 23.5.5.4 enum \_flexio\_i2s\_status\_flags

Enumerator

*kFLEXIO\_I2S\_TxDataRegEmptyFlag* Transmit buffer empty flag.  
*kFLEXIO\_I2S\_RxDataRegFullFlag* Receive buffer full flag.

### 23.5.5.5 enum flexio\_i2s\_sample\_rate\_t

Enumerator

*kFLEXIO\_I2S\_SampleRate8KHz* Sample rate 8000Hz.  
*kFLEXIO\_I2S\_SampleRate11025Hz* Sample rate 11025Hz.  
*kFLEXIO\_I2S\_SampleRate12KHz* Sample rate 12000Hz.  
*kFLEXIO\_I2S\_SampleRate16KHz* Sample rate 16000Hz.  
*kFLEXIO\_I2S\_SampleRate22050Hz* Sample rate 22050Hz.  
*kFLEXIO\_I2S\_SampleRate24KHz* Sample rate 24000Hz.  
*kFLEXIO\_I2S\_SampleRate32KHz* Sample rate 32000Hz.  
*kFLEXIO\_I2S\_SampleRate44100Hz* Sample rate 44100Hz.  
*kFLEXIO\_I2S\_SampleRate48KHz* Sample rate 48000Hz.  
*kFLEXIO\_I2S\_SampleRate96KHz* Sample rate 96000Hz.

### 23.5.5.6 enum flexio\_i2s\_word\_width\_t

Enumerator

*kFLEXIO\_I2S\_WordWidth8bits* Audio data width 8 bits.  
*kFLEXIO\_I2S\_WordWidth16bits* Audio data width 16 bits.  
*kFLEXIO\_I2S\_WordWidth24bits* Audio data width 24 bits.  
*kFLEXIO\_I2S\_WordWidth32bits* Audio data width 32 bits.

## 23.5.6 Function Documentation

### 23.5.6.1 void FLEXIO\_I2S\_Init ( FLEXIO\_I2S\_Type \* *base*, const flexio\_i2s\_config\_t \* *config* )

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by [FLEXIO\\_I2S\\_GetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | FlexIO I2S base pointer         |
| <i>config</i> | FlexIO I2S configure structure. |

### 23.5.6.2 void FLEXIO\_I2S\_GetDefaultConfig ( flexio\_i2s\_config\_t \* *config* )

The purpose of this API is to get the configuration structure initialized for use in [FLEXIO\\_I2S\\_Init\(\)](#). Users may use the initialized structure unchanged in [FLEXIO\\_I2S\\_Init\(\)](#) or modify some fields of the structure before calling [FLEXIO\\_I2S\\_Init\(\)](#).

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | pointer to master configuration structure |
|---------------|-------------------------------------------|

### 23.5.6.3 void FLEXIO\_I2S\_Deinit ( FLEXIO\_I2S\_Type \* *base* )

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the [FLEXIO\\_I2S\\_Init](#) to use the FlexIO I2S module.

## FlexIO I2S Driver

Parameters

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

**23.5.6.4 static void FLEXIO\_I2S\_Enable ( FLEXIO\_I2S\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a>      |
| <i>enable</i> | True to enable, false dose not have any effect. |

**23.5.6.5 uint32\_t FLEXIO\_I2S\_GetStatusFlags ( FLEXIO\_I2S\_Type \* *base* )**

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
|-------------|------------------------------------------------------|

Returns

Status flag, which are ORed by the enumerators in the `_flexio_i2s_status_flags`.

**23.5.6.6 void FLEXIO\_I2S\_EnableInterrupts ( FLEXIO\_I2S\_Type \* *base*, uint32\_t *mask* )**

This function enables the FlexIO UART interrupt.

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>mask</i> | interrupt source                                     |

**23.5.6.7 void FLEXIO\_I2S\_DisableInterrupts ( FLEXIO\_I2S\_Type \* *base*, uint32\_t *mask* )**

This function enables the FlexIO UART interrupt.

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>mask</i> | interrupt source                                     |

**23.5.6.8** `static void FLEXIO_I2S_TxEnableDMA ( FLEXIO_I2S_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FlexIO I2S base pointer                         |
| <i>enable</i> | True means enable DMA, false means disable DMA. |

**23.5.6.9** `static void FLEXIO_I2S_RxEnableDMA ( FLEXIO_I2S_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FlexIO I2S base pointer                         |
| <i>enable</i> | True means enable DMA, false means disable DMA. |

**23.5.6.10** `static uint32_t FLEXIO_I2S_TxGetDataRegisterAddress ( FLEXIO_I2S_Type * base ) [inline], [static]`

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
|-------------|------------------------------------------------------|

Returns

FlexIO i2s send data register address.

**23.5.6.11** `static uint32_t FLEXIO_I2S_RxGetDataRegisterAddress ( FLEXIO_I2S_Type * base ) [inline], [static]`

This function returns the I2S data register address, mainly used by DMA/eDMA.

## FlexIO I2S Driver

### Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
|-------------|------------------------------------------------------|

### Returns

FlexIO i2s receive data register address.

#### 23.5.6.12 void FLEXIO\_I2S\_MasterSetFormat ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

### Parameters

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

#### 23.5.6.13 void FLEXIO\_I2S\_SlaveSetFormat ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_format\_t \* *format* )

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

### Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>format</i> | Pointer to FlexIO I2S audio data format structure.   |

#### 23.5.6.14 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.                                    |

**23.5.6.15** `static void FLEXIO_I2S_WriteData ( FLEXIO_I2S_Type * base, uint8_t bitWidth, uint32_t data ) [inline], [static]`

## Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>base</i>     | FlexIO I2S base pointer.                                |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>data</i>     | Data to be written.                                     |

**23.5.6.16** `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.                                       |

**23.5.6.17** `static uint32_t FLEXIO_I2S_ReadData ( FLEXIO_I2S_Type * base ) [inline], [static]`

## FlexIO I2S Driver

### Parameters

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

### Returns

Data read from data register.

**23.5.6.18 void FLEXIO\_I2S\_TransferTxCreateHandle ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_callback\_t *callback*, void \* *userData* )**

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

### Parameters

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure                  |
| <i>handle</i>   | Pointer to flexio_i2s_handle_t structure to store the transfer state. |
| <i>callback</i> | FlexIO I2S callback function, which is called while finished a block. |
| <i>userData</i> | User parameter for the FlexIO I2S callback.                           |

**23.5.6.19 void FLEXIO\_I2S\_TransferSetFormat ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )**

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

### Parameters

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

**23.5.6.20 void FLEXIO\_I2S\_TransferRxCreateHandle ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_callback\_t *callback*, void \* *userData* )**

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.



## Parameters

|                 |                                                                                    |
|-----------------|------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                              |
| <i>handle</i>   | Pointer to <code>flexio_i2s_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | FlexIO I2S callback function, which is called while finished a block.              |
| <i>userData</i> | User parameter for the FlexIO I2S callback.                                        |

### 23.5.6.21 `status_t FLEXIO_I2S_TransferSendNonBlocking ( FLEXIO_I2S_Type * base, flexio_i2s_handle_t * handle, flexio_i2s_transfer_t * xfer )`

## Note

The API returns immediately after transfer initiates. Call `FLEXIO_I2S_GetRemainingBytes` to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

## Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                 |
| <i>handle</i> | Pointer to <code>flexio_i2s_handle_t</code> 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.                                                    |

### 23.5.6.22 `status_t FLEXIO_I2S_TransferReceiveNonBlocking ( FLEXIO_I2S_Type * base, flexio_i2s_handle_t * handle, flexio_i2s_transfer_t * xfer )`

## Note

The API returns immediately after transfer initiates. Call `FLEXIO_I2S_GetRemainingBytes` to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

## 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 |
| <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.      |

### 23.5.6.23 void FLEXIO\_I2S\_TransferAbortSend ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle* )

#### Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

### Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                 |
| <i>handle</i> | Pointer to <code>flexio_i2s_handle_t</code> structure which stores the transfer state |

### 23.5.6.24 void FLEXIO\_I2S\_TransferAbortReceive ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle* )

#### Note

This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

### Parameters

---

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t structure which stores the transfer state |

**23.5.6.25** `status_t FLEXIO_I2S_TransferGetSendCount ( FLEXIO_I2S_Type * base, flexio_i2s_handle_t * handle, size_t * count )`

Parameters

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

**23.5.6.26** `status_t FLEXIO_I2S_TransferGetReceiveCount ( FLEXIO_I2S_Type * base, flexio_i2s_handle_t * handle, size_t * count )`

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t 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. |

**23.5.6.27** `void FLEXIO_I2S_TransferTxHandleIRQ ( void * i2sBase, void * i2sHandle )`

## FlexIO I2S Driver

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>i2sBase</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure. |
| <i>i2sHandle</i> | Pointer to flexio_i2s_handle_t structure              |

### 23.5.6.28 void FLEXIO\_I2S\_TransferRxHandleIRQ ( void \* *i2sBase*, void \* *i2sHandle* )

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>i2sBase</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure. |
| <i>i2sHandle</i> | Pointer to flexio_i2s_handle_t structure.             |

## 23.5.7 FlexIO eDMA I2S Driver

### 23.5.7.1 Overview

#### Data Structures

- struct `flexio_i2s_edma_handle_t`  
*FlexIO I2S DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* `flexio_i2s_edma_callback_t`)(`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `status_t` status, void \*userData)  
*FlexIO I2S eDMA transfer callback function for finish and error.*

#### Driver version

- #define `FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 4))  
*FlexIO I2S EDMA driver version 2.1.4.*

#### 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.*

## FlexIO I2S Driver

- status\_t [FLEXIO\\_I2S\\_TransferGetSendCountEDMA](#) (FLEXIO\_I2S\_Type \*base, flexio\_i2s\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the remaining bytes to be sent.*
- status\_t [FLEXIO\\_I2S\\_TransferGetReceiveCountEDMA](#) (FLEXIO\_I2S\_Type \*base, flexio\_i2s\_edma\_handle\_t \*handle, size\_t \*count)  
*Get the remaining bytes to be received.*

### 23.5.7.2 Data Structure Documentation

#### 23.5.7.2.1 struct flexio\_i2s\_edma\_handle

##### Data Fields

- [edma\\_handle\\_t](#) \* [dmaHandle](#)  
*DMA handler for FlexIO I2S send.*
- uint8\_t [bytesPerFrame](#)  
*Bytes in a frame.*
- uint8\_t [nbytes](#)  
*eDMA minor byte transfer count initially configured.*
- uint32\_t [state](#)  
*Internal state for FlexIO I2S eDMA transfer.*
- [flexio\\_i2s\\_edma\\_callback\\_t](#) [callback](#)  
*Callback for users while transfer finish or error occurred.*
- void \* [userData](#)  
*User callback parameter.*
- [edma\\_tcd\\_t](#) [tcd](#) [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE+1U]  
*TCD pool for eDMA transfer.*
- [flexio\\_i2s\\_transfer\\_t](#) [queue](#) [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*

### 23.5.7.2.1.1 Field Documentation

23.5.7.2.1.1.1 `uint8_t flexio_i2s_edma_handle_t::nbytes`

23.5.7.2.1.1.2 `edma_tcd_t flexio_i2s_edma_handle_t::tcd[FLEXIO_I2S_XFER_QUEUE_SIZE+1U]`

23.5.7.2.1.1.3 `flexio_i2s_transfer_t flexio_i2s_edma_handle_t::queue[FLEXIO_I2S_XFER_QUEUE_SIZE]`

23.5.7.2.1.1.4 `volatile uint8_t flexio_i2s_edma_handle_t::queueUser`

### 23.5.7.3 Macro Definition Documentation

23.5.7.3.1 `#define FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))`

### 23.5.7.4 Function Documentation

23.5.7.4.1 `void FLEXIO_I2S_TransferTxCreateHandleEDMA ( FLEXIO_I2S_Type * base, flexio_i2s_edma_handle_t * handle, flexio_i2s_edma_callback_t callback, void * userData, edma_handle_t * dmaHandle )`

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

|                  |                                                                               |
|------------------|-------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                           |
| <i>handle</i>    | FlexIO I2S eDMA handle pointer.                                               |
| <i>callback</i>  | FlexIO I2S eDMA callback function called while finished a block.              |
| <i>userData</i>  | User parameter for callback.                                                  |
| <i>dmaHandle</i> | eDMA handle for FlexIO I2S. This handle is a static value allocated by users. |

23.5.7.4.2 `void FLEXIO_I2S_TransferRxCreateHandleEDMA ( FLEXIO_I2S_Type * base, flexio_i2s_edma_handle_t * handle, flexio_i2s_edma_callback_t callback, void * userData, edma_handle_t * dmaHandle )`

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

## FlexIO I2S Driver

### Parameters

|                  |                                                                               |
|------------------|-------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                           |
| <i>handle</i>    | FlexIO I2S eDMA handle pointer.                                               |
| <i>callback</i>  | FlexIO I2S eDMA callback function called while finished a block.              |
| <i>userData</i>  | User parameter for callback.                                                  |
| <i>dmaHandle</i> | eDMA handle for FlexIO I2S. This handle is a static value allocated by users. |

**23.5.7.4.3 void FLEXIO\_I2S\_TransferSetFormatEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )**

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

### Parameters

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <i>base</i>        | FlexIO I2S peripheral base address.                                          |
| <i>handle</i>      | FlexIO I2S eDMA handle pointer                                               |
| <i>format</i>      | Pointer to FlexIO I2S audio data format structure.                           |
| <i>srcClock_Hz</i> | FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode. |

### Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully.  |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid. |

**23.5.7.4.4 status\_t FLEXIO\_I2S\_TransferSendEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_transfer\_t \* *xfer* )**

### Note

This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

### Parameters



|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

## Return values

|                                |                                            |
|--------------------------------|--------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S eDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.            |
| <i>kStatus_TxBusy</i>          | FlexIO I2S is busy sending data.           |

#### 23.5.7.4.5 status\_t FLEXIO\_I2S\_TransferReceiveEDMA ( FLEXIO\_I2S\_Type \* base, flexio\_i2s\_edma\_handle\_t \* handle, flexio\_i2s\_transfer\_t \* xfer )

## Note

This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

## Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S eDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.               |
| <i>kStatus_RxBusy</i>          | FlexIO I2S is busy receiving data.            |

#### 23.5.7.4.6 void FLEXIO\_I2S\_TransferAbortSendEDMA ( FLEXIO\_I2S\_Type \* base, flexio\_i2s\_edma\_handle\_t \* handle )

## Parameters

## FlexIO I2S Driver

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

**23.5.7.4.7 void FLEXIO\_I2S\_TransferAbortReceiveEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle* )**

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

**23.5.7.4.8 status\_t FLEXIO\_I2S\_TransferGetSendCountEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>count</i>  | Bytes sent.                         |

Return values

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

**23.5.7.4.9 status\_t FLEXIO\_I2S\_TransferGetReceiveCountEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

|              |                 |
|--------------|-----------------|
| <i>count</i> | Bytes received. |
|--------------|-----------------|

## Return values

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

## FlexIO I2S Driver

### 23.5.8 FlexIO DMA I2S Driver

#### 23.5.8.1 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.*

#### Driver version

- #define `FSL_FLEXIO_I2S_DMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 4))  
*FlexIO I2S DMA driver version 2.1.4.*

#### 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.*

## 23.5.8.2 Data Structure Documentation

### 23.5.8.2.1 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.*

## FlexIO I2S Driver

### 23.5.8.2.1.1 Field Documentation

23.5.8.2.1.1.1 `flexio_i2s_transfer_t flexio_i2s_dma_handle_t::queue[FLEXIO_I2S_XFER_QUEUE_SIZE]`

23.5.8.2.1.1.2 `volatile uint8_t flexio_i2s_dma_handle_t::queueUser`

### 23.5.8.3 Macro Definition Documentation

23.5.8.3.1 `#define FSL_FLEXIO_I2S_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))`

### 23.5.8.4 Function Documentation

23.5.8.4.1 `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, 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 is a static value allocated by users. |

23.5.8.4.2 `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, 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 is a static value allocated by users. |

**23.5.8.4.3 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 at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the DMA parameter according to the 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.                           |
| <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. |

**23.5.8.4.4 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

## FlexIO I2S Driver

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

#### 23.5.8.4.5 **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.           |

#### 23.5.8.4.6 **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.      |

**23.5.8.4.7 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.      |

**23.5.8.4.8 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.      |
| <i>count</i>  | Bytes sent.                         |

Return values

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

**23.5.8.4.9 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.      |

## FlexIO I2S Driver

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

## 23.6 FlexIO MCU Interface LCD Driver

### 23.6.1 Overview

The MCUXpresso SDK provides a peripheral driver for LCD (8080 or 6800 interface) function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO LCD driver supports both 8-bit and 16-bit data bus, 8080 and 6800 interface. User could change the macro `FLEXIO_MCULCD_DATA_BUS_WIDTH` to choose 8-bit data bus or 16-bit data bus.

The FlexIO LCD driver supports three kinds of data transfer:

1. Send a data array. For example, send the LCD image data to the LCD controller.
2. Send a value many times. For example, send 0 many times to clean the LCD screen.
3. Read data into a data array. For example, read image from LCD controller.

The FlexIO LCD driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FlexIO LCD initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO LCD peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO LCD functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code.

Transactional APIs support asynchronous transfer. This means that the function `FLEXIO_MCULCD_-TransferNonBlocking` sets up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXIO_MCULCD_Idle` status.

### 23.6.2 Typical use case

#### 23.6.2.1 FlexIO LCD send/receive using functional APIs

This example shows how to send command, or write and read data using the functional APIs. The data bus is 16-bit.

```
uint16_t dataToSend[] = { ... };
uint16_t dataToReceive[] = { ... };

FLEXIO_MCULCD_Type flexioLcdDev;
flexio_MCULCD_transfer_t xfer;
flexio_MCULCD_config_t config;

FLEXIO_MCULCD_GetDefaultConfig(&config);
FLEXIO_MCULCD_Init(&flexioLcdDev, &config, 120000000);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
```

## FlexIO MCU Interface LCD Driver

```
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command1);
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command1;
xfer.dataCount = 0; // Only send command, no data transfer.
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteDataArrayBlocking(&flexioLcdDev, dataToSend, sizeof(
    dataToSend));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
xfer.dataCount = sizeof(dataToSend);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteSameValueBlocking(&flexioLcdDev, value, 1000); //
    Send value 1000 times
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
xfer.dataAddrOrSameValue = value;
xfer.dataCount = 1000;
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command3);
FLEXIO_MCULCD_ReadDataArrayBlocking(&flexioLcdDev, dataToReceive, sizeof(
    dataToReceive));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);
```

### 23.6.2.2 FlexIO LCD send/receive using interrupt transactional APIs

```
flexio_MCULCD_handle_t handle;
volatile bool completeFlag = false;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_handle_t *handle, status_t
    status, void *userData)
{
    if (kStatus_FLEXIO_MCULCD_Idle == status)
    {
        completeFlag = true;
    }
}

void main(void)
```

```

{
    // Init the FlexIO LCD driver.
    FLEXIO_MCULCD_Init(...);

    // Create the transactional handle.
    FLEXIO_MCULCD_TransferCreateHandle(&flexioLcdDev, &handle,
        flexioLcdCallback, NULL);

    xfer.command = command1;
    xfer.dataCount = 0; // Only send command, no data transfer.
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &xfer);

    // When only send method, it is not necessary to wait for the callback,
    // because the command is sent using a blocking method internally. The
    // command has been sent out after the function FLEXIO_MCULCD_TransferNonBlocking
    // returns.
    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
    xfer.dataCount = sizeof(dataToSend);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
    xfer.dataAddrOrSameValue = value;
    xfer.dataCount = 1000;
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command3;
    xfer.mode = kFLEXIO_MCULCD_ReadArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
    xfer.dataCount = sizeof(dataToReceive);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }
}

```

## Modules

- [FlexIO eDMA MCU Interface LCD Driver](#)

*SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.*

### Data Structures

- struct [FLEXIO\\_MCULCD\\_Type](#)  
*Define FlexIO MCULCD access structure typedef. [More...](#)*
- struct [flexio\\_mculcd\\_config\\_t](#)  
*Define FlexIO MCULCD configuration structure. [More...](#)*
- struct [flexio\\_mculcd\\_transfer\\_t](#)  
*Define FlexIO MCULCD transfer structure. [More...](#)*
- struct [flexio\\_mculcd\\_handle\\_t](#)  
*Define FlexIO MCULCD handle structure. [More...](#)*

### Macros

- #define [FLEXIO\\_MCULCD\\_WAIT\\_COMPLETE\\_TIME](#) 512  
*The delay time to wait for FLEXIO transmit complete.*
- #define [FLEXIO\\_MCULCD\\_DATA\\_BUS\\_WIDTH](#) 16  
*The data bus width, must be 8 or 16.*

### Typedefs

- typedef void(\* [flexio\\_mculcd\\_pin\\_func\\_t](#))(bool set)  
*Function to set or clear the CS and RS pin.*
- typedef void(\* [flexio\\_mculcd\\_transfer\\_callback\\_t](#))([FLEXIO\\_MCULCD\\_Type](#) \*base, [flexio\\_mculcd\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
*FlexIO MCULCD callback for finished transfer.*

### Enumerations

- enum [\\_flexio\\_mculcd\\_status](#) {  
[kStatus\\_FLEXIO\\_MCULCD\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_MCULCD, 0),  
[kStatus\\_FLEXIO\\_MCULCD\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_MCULCD, 1),  
[kStatus\\_FLEXIO\\_MCULCD\\_Error](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_MCULCD, 2) }  
*FlexIO LCD transfer status.*
- enum [flexio\\_mculcd\\_bus\\_t](#) {  
[kFLEXIO\\_MCULCD\\_8080](#),  
[kFLEXIO\\_MCULCD\\_6800](#) }  
*Define FlexIO MCULCD bus type.*
- enum [\\_flexio\\_mculcd\\_interrupt\\_enable](#) {  
[kFLEXIO\\_MCULCD\\_TxEmptyInterruptEnable](#) = (1U << 0U),  
[kFLEXIO\\_MCULCD\\_RxFullInterruptEnable](#) = (1U << 1U) }  
*Define FlexIO MCULCD interrupt mask.*
- enum [\\_flexio\\_mculcd\\_status\\_flags](#) {  
[kFLEXIO\\_MCULCD\\_TxEmptyFlag](#) = (1U << 0U),  
[kFLEXIO\\_MCULCD\\_RxFullFlag](#) = (1U << 1U) }  
*Define FlexIO MCULCD status mask.*

- enum `_flexio_mculcd_dma_enable` {  
`kFLEXIO_MCULCD_TxDmaEnable` = 0x1U,  
`kFLEXIO_MCULCD_RxDmaEnable` = 0x2U }  
*Define FlexIO MCULCD DMA mask.*
- enum `flexio_mculcd_transfer_mode_t` {  
`kFLEXIO_MCULCD_ReadArray`,  
`kFLEXIO_MCULCD_WriteArray`,  
`kFLEXIO_MCULCD_WriteSameValue` }  
*Transfer mode.*

## Driver version

- #define `FSL_FLEXIO_MCULCD_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*FlexIO MCULCD driver version 2.0.2.*

## FlexIO MCULCD Configuration

- status\_t `FLEXIO_MCULCD_Init` (`FLEXIO_MCULCD_Type` \*base, `flexio_mculcd_config_t` \*config, uint32\_t srcClock\_Hz)  
*Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with FlexIO MCULCD configuration.*
- void `FLEXIO_MCULCD_Deinit` (`FLEXIO_MCULCD_Type` \*base)  
*Resets the FLEXIO\_MCULCD timer and shifter configuration.*
- void `FLEXIO_MCULCD_GetDefaultConfig` (`flexio_mculcd_config_t` \*config)  
*Gets the default configuration to configure the FlexIO MCULCD.*

## Status

- uint32\_t `FLEXIO_MCULCD_GetStatusFlags` (`FLEXIO_MCULCD_Type` \*base)  
*Gets FlexIO MCULCD status flags.*
- void `FLEXIO_MCULCD_ClearStatusFlags` (`FLEXIO_MCULCD_Type` \*base, uint32\_t mask)  
*Clears FlexIO MCULCD status flags.*

## Interrupts

- void `FLEXIO_MCULCD_EnableInterrupts` (`FLEXIO_MCULCD_Type` \*base, uint32\_t mask)  
*Enables the FlexIO MCULCD interrupt.*
- void `FLEXIO_MCULCD_DisableInterrupts` (`FLEXIO_MCULCD_Type` \*base, uint32\_t mask)  
*Disables the FlexIO MCULCD interrupt.*

## DMA Control

- static void `FLEXIO_MCULCD_EnableTxDMA` (`FLEXIO_MCULCD_Type` \*base, bool enable)

## FlexIO MCU Interface LCD Driver

- *Enables/disables the FlexIO MCULCD transmit DMA.*  
static void [FLEXIO\\_MCULCD\\_EnableRxDMA](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, bool enable)
- *Enables/disables the FlexIO MCULCD receive DMA.*  
static uint32\_t [FLEXIO\\_MCULCD\\_GetTxDataRegisterAddress](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)
- *Gets the FlexIO MCULCD transmit data register address.*  
static uint32\_t [FLEXIO\\_MCULCD\\_GetRxDataRegisterAddress](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)
- *Gets the FlexIO MCULCD receive data register address.*

## Bus Operations

- status\_t [FLEXIO\\_MCULCD\\_SetBaudRate](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, uint32\_t baudRate-\_Bps, uint32\_t srcClock\_Hz)  
*Set desired baud rate.*
- void [FLEXIO\\_MCULCD\\_SetSingleBeatWriteConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Configures the FLEXIO MCULCD to multiple beats write mode.*
- void [FLEXIO\\_MCULCD\\_ClearSingleBeatWriteConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Clear the FLEXIO MCULCD multiple beats write mode configuration.*
- void [FLEXIO\\_MCULCD\\_SetSingleBeatReadConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Configures the FLEXIO MCULCD to multiple beats read mode.*
- void [FLEXIO\\_MCULCD\\_ClearSingleBeatReadConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Clear the FLEXIO MCULCD multiple beats read mode configuration.*
- void [FLEXIO\\_MCULCD\\_SetMultiBeatsWriteConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Configures the FLEXIO MCULCD to multiple beats write mode.*
- void [FLEXIO\\_MCULCD\\_ClearMultiBeatsWriteConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Clear the FLEXIO MCULCD multiple beats write mode configuration.*
- void [FLEXIO\\_MCULCD\\_SetMultiBeatsReadConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Configures the FLEXIO MCULCD to multiple beats read mode.*
- void [FLEXIO\\_MCULCD\\_ClearMultiBeatsReadConfig](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Clear the FLEXIO MCULCD multiple beats read mode configuration.*
- static void [FLEXIO\\_MCULCD\\_Enable](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, bool enable)  
*Enables/disables the FlexIO MCULCD module operation.*
- uint32\_t [FLEXIO\\_MCULCD\\_ReadData](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Read data from the FLEXIO MCULCD RX shifter buffer.*
- static void [FLEXIO\\_MCULCD\\_WriteData](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, uint32\_t data)  
*Write data into the FLEXIO MCULCD TX shifter buffer.*
- static void [FLEXIO\\_MCULCD\\_StartTransfer](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*Assert the nCS to start transfer.*
- static void [FLEXIO\\_MCULCD\\_StopTransfer](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base)  
*De-assert the nCS to stop transfer.*
- void [FLEXIO\\_MCULCD\\_WaitTransmitComplete](#) (void)  
*Wait for transmit data send out finished.*
- void [FLEXIO\\_MCULCD\\_WriteCommandBlocking](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, uint32\_t command)  
*Send command in blocking way.*
- void [FLEXIO\\_MCULCD\\_WriteDataArrayBlocking](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, void \*data, size\_t size)  
*Send data array in blocking way.*



- void `FLEXIO_MCULCD_ReadDataArrayBlocking` (`FLEXIO_MCULCD_Type *base`, void `*data`, `size_t size`)  
*Read data into array in blocking way.*
- void `FLEXIO_MCULCD_WriteSameValueBlocking` (`FLEXIO_MCULCD_Type *base`, `uint32_t sameValue`, `size_t size`)  
*Send the same value many times in blocking way.*
- void `FLEXIO_MCULCD_TransferBlocking` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_transfer_t *xfer`)  
*Performs a polling transfer.*

## Transactional

- `status_t FLEXIO_MCULCD_TransferCreateHandle` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`, `flexio_mculcd_transfer_callback_t callback`, void `*userData`)  
*Initializes the FlexIO MCULCD handle, which is used in transactional functions.*
- `status_t FLEXIO_MCULCD_TransferNonBlocking` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`, `flexio_mculcd_transfer_t *xfer`)  
*Transfer data using IRQ.*
- void `FLEXIO_MCULCD_TransferAbort` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`)  
*Aborts the data transfer, which used IRQ.*
- `status_t FLEXIO_MCULCD_TransferGetCount` (`FLEXIO_MCULCD_Type *base`, `flexio_mculcd_handle_t *handle`, `size_t *count`)  
*Gets the data transfer status which used IRQ.*
- void `FLEXIO_MCULCD_TransferHandleIRQ` (void `*base`, void `*handle`)  
*FlexIO MCULCD IRQ handler function.*

## 23.6.3 Data Structure Documentation

### 23.6.3.1 struct FLEXIO\_MCULCD\_Type

#### Data Fields

- `FLEXIO_Type * flexioBase`  
*FlexIO base pointer.*
- `flexio_mculcd_bus_t busType`  
*The bus type, 8080 or 6800.*
- `uint8_t dataPinStartIndex`  
*Start index of the data pin, the FlexIO pin `dataPinStartIndex` to  $(dataPinStartIndex + FLEXIO_MCULCD_DATA_BUS_WIDTH - 1)$  will be used for data transfer.*
- `uint8_t ENWRPinIndex`  
*Pin select for WR(8080 mode), EN(6800 mode).*
- `uint8_t RDPinIndex`  
*Pin select for RD(8080 mode), not used in 6800 mode.*
- `uint8_t txShifterStartIndex`  
*Start index of shifters used for data write, it must be 0 or 4.*
- `uint8_t txShifterEndIndex`

## FlexIO MCU Interface LCD Driver

- *End index of shifters used for data write.*  
uint8\_t [rxShifterStartIndex](#)
- *Start index of shifters used for data read.*  
uint8\_t [rxShifterEndIndex](#)
- *End index of shifters used for data read, it must be 3 or 7.*  
uint8\_t [timerIndex](#)
- *Timer index used in FlexIO MCULCD.*  
[flexio\\_mculcd\\_pin\\_func\\_t setCSPin](#)  
*Function to set or clear the CS pin.*
- [flexio\\_mculcd\\_pin\\_func\\_t setRSPin](#)  
*Function to set or clear the RS pin.*
- [flexio\\_mculcd\\_pin\\_func\\_t setRDWRPin](#)  
*Function to set or clear the RD/WR pin, only used in 6800 mode.*

### 23.6.3.1.0.1 Field Documentation

23.6.3.1.0.1.1 [FLEXIO\\_Type\\* FLEXIO\\_MCULCD\\_Type::flexioBase](#)

23.6.3.1.0.1.2 [flexio\\_mculcd\\_bus\\_t FLEXIO\\_MCULCD\\_Type::busType](#)

23.6.3.1.0.1.3 [uint8\\_t FLEXIO\\_MCULCD\\_Type::dataPinStartIndex](#)

Only support data bus width 8 and 16.

23.6.3.1.0.1.4 [uint8\\_t FLEXIO\\_MCULCD\\_Type::ENWRPinIndex](#)

23.6.3.1.0.1.5 [uint8\\_t FLEXIO\\_MCULCD\\_Type::RDPinIndex](#)

23.6.3.1.0.1.6 [uint8\\_t FLEXIO\\_MCULCD\\_Type::txShifterStartIndex](#)

23.6.3.1.0.1.7 [uint8\\_t FLEXIO\\_MCULCD\\_Type::txShifterEndIndex](#)

23.6.3.1.0.1.8 [uint8\\_t FLEXIO\\_MCULCD\\_Type::rxShifterStartIndex](#)

23.6.3.1.0.1.9 [uint8\\_t FLEXIO\\_MCULCD\\_Type::rxShifterEndIndex](#)

23.6.3.1.0.1.10 [uint8\\_t FLEXIO\\_MCULCD\\_Type::timerIndex](#)

23.6.3.1.0.1.11 [flexio\\_mculcd\\_pin\\_func\\_t FLEXIO\\_MCULCD\\_Type::setCSPin](#)

23.6.3.1.0.1.12 [flexio\\_mculcd\\_pin\\_func\\_t FLEXIO\\_MCULCD\\_Type::setRSPin](#)

23.6.3.1.0.1.13 [flexio\\_mculcd\\_pin\\_func\\_t FLEXIO\\_MCULCD\\_Type::setRDWRPin](#)

### 23.6.3.2 struct [flexio\\_mculcd\\_config\\_t](#)

#### Data Fields

- bool [enable](#)  
*Enable/disable FlexIO MCULCD after configuration.*
- bool [enableInDoze](#)

- *Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*
- bool [enableFastAccess](#)  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud rate in Bps.*

### 23.6.3.2.0.2 Field Documentation

23.6.3.2.0.2.1 bool [flexio\\_mculcd\\_config\\_t::enable](#)

23.6.3.2.0.2.2 bool [flexio\\_mculcd\\_config\\_t::enableInDoze](#)

23.6.3.2.0.2.3 bool [flexio\\_mculcd\\_config\\_t::enableInDebug](#)

23.6.3.2.0.2.4 bool [flexio\\_mculcd\\_config\\_t::enableFastAccess](#)

23.6.3.2.0.2.5 uint32\_t [flexio\\_mculcd\\_config\\_t::baudRate\\_Bps](#)

### 23.6.3.3 struct [flexio\\_mculcd\\_transfer\\_t](#)

#### Data Fields

- uint32\_t [command](#)  
*Command to send.*
- [flexio\\_mculcd\\_transfer\\_mode\\_t](#) [mode](#)  
*Transfer mode.*
- uint32\_t [dataAddrOrSameValue](#)  
*When sending the same value for many times, this is the value to send.*
- size\_t [dataSize](#)  
*How many bytes to transfer.*

### 23.6.3.3.0.3 Field Documentation

23.6.3.3.0.3.1 uint32\_t [flexio\\_mculcd\\_transfer\\_t::command](#)

23.6.3.3.0.3.2 [flexio\\_mculcd\\_transfer\\_mode\\_t](#) [flexio\\_mculcd\\_transfer\\_t::mode](#)

23.6.3.3.0.3.3 uint32\_t [flexio\\_mculcd\\_transfer\\_t::dataAddrOrSameValue](#)

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

23.6.3.3.0.3.4 size\_t [flexio\\_mculcd\\_transfer\\_t::dataSize](#)

### 23.6.3.4 struct [\\_flexio\\_mculcd\\_handle](#)

typedef for [flexio\\_mculcd\\_handle\\_t](#) in advance.

## FlexIO MCU Interface LCD Driver

### Data Fields

- uint32\_t [dataAddrOrSameValue](#)  
*When sending the same value for many times, this is the value to send.*
- size\_t [dataCount](#)  
*Total count to be transferred.*
- volatile size\_t [remainingCount](#)  
*Remaining count to transfer.*
- volatile uint32\_t [state](#)  
*FlexIO MCULCD internal state.*
- [flexio\\_mculcd\\_transfer\\_callback\\_t](#) [completionCallback](#)  
*FlexIO MCULCD transfer completed callback.*
- void \* [userData](#)  
*Callback parameter.*

#### 23.6.3.4.0.4 Field Documentation

##### 23.6.3.4.0.4.1 uint32\_t flexio\_mculcd\_handle\_t::dataAddrOrSameValue

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

##### 23.6.3.4.0.4.2 size\_t flexio\_mculcd\_handle\_t::dataCount

##### 23.6.3.4.0.4.3 volatile size\_t flexio\_mculcd\_handle\_t::remainingCount

##### 23.6.3.4.0.4.4 volatile uint32\_t flexio\_mculcd\_handle\_t::state

##### 23.6.3.4.0.4.5 flexio\_mculcd\_transfer\_callback\_t flexio\_mculcd\_handle\_t::completionCallback

##### 23.6.3.4.0.4.6 void\* flexio\_mculcd\_handle\_t::userData

### 23.6.4 Macro Definition Documentation

#### 23.6.4.1 #define FSL\_FLEXIO\_MCULCD\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

#### 23.6.4.2 #define FLEXIO\_MCULCD\_WAIT\_COMPLETE\_TIME 512

Currently there is no method to detect whether the data has been sent out from the shifter, so the driver use a software delay for this. When the data is written to shifter buffer, the driver call the delay function to wait for the data shift out. If this value is too small, then the last few bytes might be lost when writing data using interrupt method or DMA method.

## 23.6.5 Typedef Documentation

**23.6.5.1** typedef void(\* flexio\_mculcd\_pin\_func\_t)(bool set)

**23.6.5.2** typedef void(\* flexio\_mculcd\_transfer\_callback\_t)(FLEXIO\_MCULCD\_Type \*base, flexio\_mculcd\_handle\_t \*handle, status\_t status, void \*userData)

When transfer finished, the callback function is called and returns the `status` as `kStatus_FLEXIO_MCULCD_Idle`.

## 23.6.6 Enumeration Type Documentation

**23.6.6.1** enum flexio\_mculcd\_status

Enumerator

*kStatus\_FLEXIO\_MCULCD\_Idle* FlexIO LCD is idle.  
*kStatus\_FLEXIO\_MCULCD\_Busy* FlexIO LCD is busy.  
*kStatus\_FLEXIO\_MCULCD\_Error* FlexIO LCD error occurred.

**23.6.6.2** enum flexio\_mculcd\_bus\_t

Enumerator

*kFLEXIO\_MCULCD\_8080* Using Intel 8080 bus.  
*kFLEXIO\_MCULCD\_6800* Using Motorola 6800 bus.

**23.6.6.3** enum flexio\_mculcd\_interrupt\_enable

Enumerator

*kFLEXIO\_MCULCD\_TxEmptyInterruptEnable* Transmit buffer empty interrupt enable.  
*kFLEXIO\_MCULCD\_RxFullInterruptEnable* Receive buffer full interrupt enable.

**23.6.6.4** enum flexio\_mculcd\_status\_flags

Enumerator

*kFLEXIO\_MCULCD\_TxEmptyFlag* Transmit buffer empty flag.  
*kFLEXIO\_MCULCD\_RxFullFlag* Receive buffer full flag.

## FlexIO MCU Interface LCD Driver

### 23.6.6.5 enum `_flexio_mculcd_dma_enable`

Enumerator

- kFLEXIO\_MCULCD\_TxDmaEnable* Tx DMA request source.
- kFLEXIO\_MCULCD\_RxDmaEnable* Rx DMA request source.

### 23.6.6.6 enum `flexio_mculcd_transfer_mode_t`

Enumerator

- kFLEXIO\_MCULCD\_ReadArray* Read data into an array.
- kFLEXIO\_MCULCD\_WriteArray* Write data from an array.
- kFLEXIO\_MCULCD\_WriteSameValue* Write the same value many times.

## 23.6.7 Function Documentation

### 23.6.7.1 `status_t FLEXIO_MCULCD_Init ( FLEXIO_MCULCD_Type * base, flexio_mculcd_config_t * config, uint32_t srcClock_Hz )`

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_MCULCD\\_GetDefaultConfig](#).

Parameters

|                    |                                                                  |
|--------------------|------------------------------------------------------------------|
| <i>base</i>        | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.     |
| <i>config</i>      | Pointer to the <a href="#">flexio_mculcd_config_t</a> structure. |
| <i>srcClock_Hz</i> | FlexIO source clock in Hz.                                       |

Return values

|                                |                                                    |
|--------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>         | Initialization success.                            |
| <i>kStatus_InvalidArgument</i> | Initialization failed because of invalid argument. |

### 23.6.7.2 `void FLEXIO_MCULCD_Deinit ( FLEXIO_MCULCD_Type * base )`

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

### 23.6.7.3 void FLEXIO\_MCULCD\_GetDefaultConfig ( flexio\_mculcd\_config\_t \* config )

The default configuration value is:

```
* config->enable = true;
* config->enableInDoze = false;
* config->enableInDebug = true;
* config->enableFastAccess = true;
* config->baudRate_Bps = 96000000U;
*
```

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>Config</i> | Pointer to the <a href="#">flexio_mculcd_config_t</a> structure. |
|---------------|------------------------------------------------------------------|

### 23.6.7.4 uint32\_t FLEXIO\_MCULCD\_GetStatusFlags ( FLEXIO\_MCULCD\_Type \* base )

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

Returns

status flag; OR'ed value or the [\\_flexio\\_mculcd\\_status\\_flags](#).

Note

Don't use this function with DMA APIs.

### 23.6.7.5 void FLEXIO\_MCULCD\_ClearStatusFlags ( FLEXIO\_MCULCD\_Type \* base, uint32\_t mask )

Parameters

## FlexIO MCU Interface LCD Driver

|             |                                                                                         |
|-------------|-----------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                            |
| <i>mask</i> | Status to clear, it is the OR'ed value of <a href="#">_flexio_mculcd_status_flags</a> . |

Note

Don't use this function with DMA APIs.

### 23.6.7.6 void FLEXIO\_MCULCD\_EnableInterrupts ( FLEXIO\_MCULCD\_Type \* *base*, uint32\_t *mask* )

This function enables the FlexIO MCULCD interrupt.

Parameters

|             |                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                                     |
| <i>mask</i> | Interrupts to enable, it is the OR'ed value of <a href="#">_flexio_mculcd_interrupt_enable</a> . |

### 23.6.7.7 void FLEXIO\_MCULCD\_DisableInterrupts ( FLEXIO\_MCULCD\_Type \* *base*, uint32\_t *mask* )

This function disables the FlexIO MCULCD interrupt.

Parameters

|             |                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                                      |
| <i>mask</i> | Interrupts to disable, it is the OR'ed value of <a href="#">_flexio_mculcd_interrupt_enable</a> . |

### 23.6.7.8 static void FLEXIO\_MCULCD\_EnableTxDMA ( FLEXIO\_MCULCD\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>mask</i> | MCULCD DMA source.                                           |



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

**23.6.7.9 static void FLEXIO\_MCULCD\_EnableRxDMA ( FLEXIO\_MCULCD\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>mask</i>   | MCULCD DMA source.                                           |
| <i>enable</i> | True means enable DMA, false means disable DMA.              |

**23.6.7.10 static uint32\_t FLEXIO\_MCULCD\_GetTxDataRegisterAddress ( FLEXIO\_MCULCD\_Type \* *base* ) [inline], [static]**

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

Returns

FlexIO MCULCD transmit data register address.

**23.6.7.11 static uint32\_t FLEXIO\_MCULCD\_GetRxDataRegisterAddress ( FLEXIO\_MCULCD\_Type \* *base* ) [inline], [static]**

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

Returns

FlexIO MCULCD receive data register address.

**23.6.7.12 status\_t FLEXIO\_MCULCD\_SetBaudRate ( FLEXIO\_MCULCD\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )**

## FlexIO MCU Interface LCD Driver

### Parameters

|                     |                                                              |
|---------------------|--------------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>baudRate_Bps</i> | Desired baud rate.                                           |
| <i>srcClock_Hz</i>  | FLEXIO clock frequency in Hz.                                |

### Return values

|                                |                              |
|--------------------------------|------------------------------|
| <i>kStatus_Success</i>         | Set successfully.            |
| <i>kStatus_InvalidArgument</i> | Could not set the baud rate. |

### 23.6.7.13 void FLEXIO\_MCULCD\_SetSingleBeatWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the beginning multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by [FLEXIO\\_MCULCD\\_ClearSingleBeatWriteConfig](#).

### Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

### Note

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

### 23.6.7.14 void FLEXIO\_MCULCD\_ClearSingleBeatWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )

Clear the write configuration set by [FLEXIO\\_MCULCD\\_SetSingleBeatWriteConfig](#).

### Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

### Note

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

### 23.6.7.15 void FLEXIO\_MCULCD\_SetSingleBeatReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the beginning or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by [FLEXIO\\_MCULCD\\_ClearSingleBeatReadConfig](#).

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

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

### 23.6.7.16 void FLEXIO\_MCULCD\_ClearSingleBeatReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

Clear the read configuration set by [FLEXIO\\_MCULCD\\_SetSingleBeatReadConfig](#).

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

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

### 23.6.7.17 void FLEXIO\_MCULCD\_SetMultiBeatsWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the beginning multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by [FLEXIO\\_MCULCD\\_ClearMultBeatsWriteConfig](#).

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

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

**23.6.7.18 void FLEXIO\_MCULCD\_ClearMultiBeatsWriteConfig ( FLEXIO\_MCULCD\_Type \* *base* )**

Clear the write configuration set by FLEXIO\_MCULCD\_SetMultBeatsWriteConfig.

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

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

### 23.6.7.19 void FLEXIO\_MCULCD\_SetMultiBeatsReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO\_MCULCD\_ClearMultBeatsReadConfig.

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

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

### 23.6.7.20 void FLEXIO\_MCULCD\_ClearMultiBeatsReadConfig ( FLEXIO\_MCULCD\_Type \* *base* )

Clear the read configuration set by FLEXIO\_MCULCD\_SetMultBeatsReadConfig.

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
|-------------|-----------------------------------------------------|

Note

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

### 23.6.7.21 static void FLEXIO\_MCULCD\_Enable ( FLEXIO\_MCULCD\_Type \* *base*, bool *enable* ) [inline], [static]

## FlexIO MCU Interface LCD Driver

### Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> . |
| <i>enable</i> | True to enable, false does not have any effect.     |

### 23.6.7.22 `uint32_t FLEXIO_MCULCD_ReadData ( FLEXIO_MCULCD_Type * base )`

Read data from the RX shift buffer directly, it does no check whether the buffer is empty or not.

If the data bus width is 8-bit:

```
* uint8_t value;  
* value = (uint8_t)FLEXIO_MCULCD_ReadData(base);  
*
```

If the data bus width is 16-bit:

```
* uint16_t value;  
* value = (uint16_t)FLEXIO_MCULCD_ReadData(base);  
*
```

### Note

This function returns the RX shifter buffer value (32-bit) directly. The return value should be converted according to data bus width.

### Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

### Returns

The data read out.

### Note

Don't use this function with DMA APIs.

### 23.6.7.23 `static void FLEXIO_MCULCD_WriteData ( FLEXIO_MCULCD_Type * base, uint32_t data ) [inline], [static]`

Write data into the TX shift buffer directly, it does no check whether the buffer is full or not.

## Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>data</i> | The data to write.                                           |

## Note

Don't use this function with DMA APIs.

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

## Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

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

## Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
|-------------|--------------------------------------------------------------|

**23.6.7.26** `void FLEXIO_MCULCD_WaitTransmitComplete ( void )`

Currently there is no effective method to wait for the data send out from the shiter, so here use a while loop to wait.

## Note

This is an internal used function.

**23.6.7.27** `void FLEXIO_MCULCD_WriteCommandBlocking ( FLEXIO_MCULCD_Type * base, uint32_t command )`

This function sends the command and returns when the command has been sent out.

## FlexIO MCU Interface LCD Driver

Parameters

|                |                                                              |
|----------------|--------------------------------------------------------------|
| <i>base</i>    | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>command</i> | The command to send.                                         |

**23.6.7.28 void FLEXIO\_MCULCD\_WriteDataArrayBlocking ( FLEXIO\_MCULCD\_Type \* *base*, void \* *data*, size\_t *size* )**

This function sends the data array and returns when the data sent out.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>data</i> | The data array to send.                                      |
| <i>size</i> | How many bytes to write.                                     |

**23.6.7.29 void FLEXIO\_MCULCD\_ReadDataArrayBlocking ( FLEXIO\_MCULCD\_Type \* *base*, void \* *data*, size\_t *size* )**

This function reads the data into array and returns when the data read finished.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>data</i> | The array to save the data.                                  |
| <i>size</i> | How many bytes to read.                                      |

**23.6.7.30 void FLEXIO\_MCULCD\_WriteSameValueBlocking ( FLEXIO\_MCULCD\_Type \* *base*, uint32\_t *sameValue*, size\_t *size* )**

This function sends the same value many times. It could be used to clear the LCD screen. If the data bus width is 8, this function will send LSB 8 bits of *sameValue* for *size* times. If the data bus is 16, this function will send LSB 16 bits of *sameValue* for *size* / 2 times.

Parameters



|                  |                                                              |
|------------------|--------------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure. |
| <i>sameValue</i> | The same value to send.                                      |
| <i>size</i>      | How many bytes to send.                                      |

**23.6.7.31 void FLEXIO\_MCULCD\_TransferBlocking ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_transfer\_t \* *xfer* )**

Note

The API does not return until the transfer finished.

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.       |
| <i>xfer</i> | pointer to <a href="#">flexio_mculcd_transfer_t</a> structure. |

**23.6.7.32 status\_t FLEXIO\_MCULCD\_TransferCreateHandle ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_handle\_t \* *handle*, flexio\_mculcd\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                                 |
| <i>handle</i>   | Pointer to the <a href="#">flexio_mculcd_handle_t</a> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                       |
| <i>userData</i> | The parameter of the callback function.                                                      |

Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.6.7.33 status\_t FLEXIO\_MCULCD\_TransferNonBlocking ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_handle\_t \* *handle*, flexio\_mculcd\_transfer\_t \* *xfer* )**

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

## FlexIO MCU Interface LCD Driver

### Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_mculcd_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | FlexIO MCULCD transfer structure. See <a href="#">flexio_mculcd_transfer_t</a> .          |

### Return values

|                                   |                                       |
|-----------------------------------|---------------------------------------|
| <i>kStatus_Success</i>            | Successfully start a transfer.        |
| <i>kStatus_InvalidArgument</i>    | Input argument is invalid.            |
| <i>kStatus_FLEXIO_MCULCD_Busy</i> | MCULCD is busy with another transfer. |

**23.6.7.34** `void FLEXIO_MCULCD_TransferAbort ( FLEXIO_MCULCD_Type * base, flexio_mculcd_handle_t * handle )`

### Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_mculcd_handle_t</code> structure to store the transfer state. |

**23.6.7.35** `status_t FLEXIO_MCULCD_TransferGetCount ( FLEXIO_MCULCD_Type * base, flexio_mculcd_handle_t * handle, size_t * count )`

### Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_mculcd_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | How many bytes transferred so far by the non-blocking transaction.                        |

### Return values

|                                     |                                         |
|-------------------------------------|-----------------------------------------|
| <i>kStatus_Success</i>              | Get the transferred count Successfully. |
| <i>kStatus_NoTransferInProgress</i> | No transfer in process.                 |

**23.6.7.36** `void FLEXIO_MCULCD_TransferHandleIRQ ( void * base, void * handle )`

## Parameters

|               |                                                                              |
|---------------|------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_MCULCD_Type</a> structure.                 |
| <i>handle</i> | Pointer to the flexio_mculcd_handle_t structure to store the transfer state. |

## FlexIO MCU Interface LCD Driver

### 23.6.8 FlexIO eDMA MCU Interface LCD Driver

SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.

Note

eDMA transactional functions use multiple beats method for better performance, in contrast, the blocking functions and interrupt functions use single beat method. The function [FLEXIO\\_MCULCD\\_ReadData](#), [FLEXIO\\_MCULCD\\_WriteData](#), [FLEXIO\\_MCULCD\\_GetStatusFlags](#), and [FLEXIO\\_MCULCD\\_ClearStatusFlags](#) are only used for single beat case, so don't use these functions to work together with eDMA functions.

```
flexio_MCULCD_edma_handle_t handle;
volatile bool completeFlag = false;
edma_handle_t rxEdmaHandle;
edma_handle_t txEdmaHandle;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_edma_handle_t *handle,
    status_t status, void *userData)
{
    if (kStatus_FLEXIO_MCULCD_Idle == status)
    {
        completeFlag = true;
    }
}

void main(void)
{
    // Create the edma Handle.
    EDMA_CreateHandle(&rxEdmaHandle, DMA0, channel);
    EDMA_CreateHandle(&txEdmaHandle, DMA0, channel);

    // Configure the DMAMUX.
    // ...
    // rxEdmaHandle should use the last FlexIO RX shifters as DMA request source.
    // txEdmaHandle should use the first FlexIO TX shifters as DMA request source.

    // Init the FlexIO LCD driver.
    FLEXIO_MCULCD_Init(...);

    // Create the transactional handle.
    FLEXIO_MCULCD_TransferCreateHandleEDMA(&flexioLcdDev, &handle,
        flexioLcdCallback, NULL, &txEdmaHandle, &rxEdmaHandle);

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
    xfer.dataCount = sizeof(dataToSend);
    completeFlag = false;
    FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
    xfer.dataAddrOrSameValue = value;
    xfer.dataCount = 1000;
    completeFlag = false;
    FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);
}
```

```
while (!completeFlag)
{
}

xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
completeFlag = false;
FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

while (!completeFlag)
{
}
}
```

### 23.7 FlexIO SPI Driver

#### 23.7.1 Overview

The MCUXpresso SDK provides a peripheral driver for an SPI function using the Flexible I/O module of MCUXpresso SDK devices.

FlexIO SPI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for FlexIO SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the [FLEXIO\\_SPI\\_Type](#) \*base as the first parameter. FlexIO SPI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the [flexio\\_spi\\_master\\_handle\\_t/flexio\\_spi\\_slave\\_handle\\_t](#) as the second parameter. Initialize the handle by calling the [FLEXIO\\_SPI\\_MasterTransferCreateHandle\(\)](#) or [FLEXIO\\_SPI\\_SlaveTransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO\\_SPI\\_MasterTransferNonBlocking\(\)/FLEXIO\\_SPI\\_SlaveTransferNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the [kStatus\\_FLEXIO\\_SPI\\_Idle](#) status.

Note that the FlexIO SPI slave driver only supports discontinuous PCS access, which is a limitation. The FlexIO SPI slave driver can support continuous PCS, but the slave cannot adapt discontinuous and continuous PCS automatically. Users can change the timer disable mode in [FLEXIO\\_SPI\\_SlaveInit](#) manually, from [kFLEXIO\\_TimerDisableOnTimerCompare](#) to [kFLEXIO\\_TimerDisableNever](#) to enable a discontinuous PCS access. Only CPHA = 0 is supported.

#### 23.7.2 Typical use case

##### 23.7.2.1 FlexIO SPI send/receive using an interrupt method

```
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];

void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle
    , status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_SPI_Idle == status)
    {
        txFinished = true;
    }
}
```

```

}

void main(void)
{
    //...
    flexio_spi_transfer_t xfer = {0};
    flexio_spi_master_config_t userConfig;

    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);
    userConfig.baudRate_Bps = 500000U;

    spiDev.flexioBase = BOARD_FLEXIO_BASE;
    spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
    spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
    spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
    spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
    spiDev.shifterIndex[0] = 0U;
    spiDev.shifterIndex[1] = 1U;
    spiDev.timerIndex[0] = 0U;
    spiDev.timerIndex[1] = 1U;

    FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

    xfer.txData = srcBuff;
    xfer.rxData = destBuff;
    xfer.dataSize = BUFFER_SIZE;
    xfer.flags = kFLEXIO_SPI_8bitMsb;
    FLEXIO_SPI_MasterTransferCreateHandle(&spiDev, &g_spiHandle,
        FLEXIO_SPI_MasterUserCallback, NULL);
    FLEXIO_SPI_MasterTransferNonBlocking(&spiDev, &g_spiHandle, &xfer);

    // Send finished.
    while (!txFinished)
    {
        // ...
    }
}

```

### 23.7.2.2 FlexIO\_SPI Send/Receive in DMA way

```

dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];
void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_dma_handle_t
    *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_SPI_Idle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    flexio_spi_transfer_t xfer = {0};
    flexio_spi_master_config_t userConfig;

    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);

```

## FlexIO SPI Driver

```
userConfig.baudRate_Bps = 500000U;

spiDev.flexioBase = BOARD_FLEXIO_BASE;
spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
spiDev.shifterIndex[0] = 0U;
spiDev.shifterIndex[1] = 1U;
spiDev.timerIndex[0] = 0U;
spiDev.timerIndex[1] = 1U;

/* Init DMAMUX. */
DMAMUX_Init(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR)

/* Init the DMA/EDMA module */
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
DMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR);
DMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL);
DMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
edma_config_t edmaConfig;

EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, &edmaConfig);
EDMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
FLEXIO_SPI_TX_DMA_CHANNEL);
EDMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
shifterIndex[0]);
dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
shifterIndex[1]);

/* Requests DMA channels for transmit and receive. */
DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_tx);
DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_rx);
DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
FLEXIO_SPI_TX_DMA_CHANNEL);
DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
FLEXIO_SPI_RX_DMA_CHANNEL);

FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

/* Initializes the buffer. */
for (i = 0; i < BUFFER_SIZE; i++)
{
    srcBuff[i] = i;
}

/* Sends to the slave. */
xfer.txData = srcBuff;
xfer.rxData = destBuff;
xfer.dataSize = BUFFER_SIZE;
xfer.flags = kFLEXIO_SPI_8bitMsb;
FLEXIO_SPI_MasterTransferCreateHandleDMA(&spiDev, &
g_spiHandle, FLEXIO_SPI_MasterUserCallback, NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);
FLEXIO_SPI_MasterTransferDMA(&spiDev, &g_spiHandle, &xfer);

// Send finished.
while (!txFinished)
{
```



```

    }
    // ...
}

```

## Modules

- [FlexIO 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...](#)*
- 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) }

## FlexIO SPI Driver

- 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, 3))  
*FlexIO SPI driver version 2.1.3.*

### FlexIO SPI Configuration

- void `FLEXIO_SPI_MasterInit` (`FLEXIO_SPI_Type` \*base, `flexio_spi_master_config_t` \*master-Config, `uint32_t` srcClock\_Hz)  
*Un gates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration.*
- void `FLEXIO_SPI_MasterDeinit` (`FLEXIO_SPI_Type` \*base)  
*Resets the FlexIO SPI timer and shifter config.*

- void [FLEXIO\\_SPI\\_MasterGetDefaultConfig](#) ([flexio\\_spi\\_master\\_config\\_t](#) \*masterConfig)  
*Gets the default configuration to configure the FlexIO SPI master.*
- void [FLEXIO\\_SPI\\_SlaveInit](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [flexio\\_spi\\_slave\\_config\\_t](#) \*slaveConfig)  
*Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration.*
- void [FLEXIO\\_SPI\\_SlaveDeinit](#) ([FLEXIO\\_SPI\\_Type](#) \*base)  
*Gates the FlexIO clock.*
- void [FLEXIO\\_SPI\\_SlaveGetDefaultConfig](#) ([flexio\\_spi\\_slave\\_config\\_t](#) \*slaveConfig)  
*Gets the default configuration to configure the FlexIO SPI slave.*

## Status

- [uint32\\_t FLEXIO\\_SPI\\_GetStatusFlags](#) ([FLEXIO\\_SPI\\_Type](#) \*base)  
*Gets FlexIO SPI status flags.*
- void [FLEXIO\\_SPI\\_ClearStatusFlags](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Clears FlexIO SPI status flags.*

## Interrupts

- void [FLEXIO\\_SPI\\_EnableInterrupts](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Enables the FlexIO SPI interrupt.*
- void [FLEXIO\\_SPI\\_DisableInterrupts](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Disables the FlexIO SPI interrupt.*

## DMA Control

- void [FLEXIO\\_SPI\\_EnableDMA](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [uint32\\_t](#) mask, bool enable)  
*Enables/disables the FlexIO SPI transmit DMA.*
- static [uint32\\_t FLEXIO\\_SPI\\_GetTxDataRegisterAddress](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [flexio\\_spi\\_shift\\_direction\\_t](#) direction)  
*Gets the FlexIO SPI transmit data register address for MSB first transfer.*
- static [uint32\\_t FLEXIO\\_SPI\\_GetRxDataRegisterAddress](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [flexio\\_spi\\_shift\\_direction\\_t](#) direction)  
*Gets the FlexIO SPI receive data register address for the MSB first transfer.*

## Bus Operations

- static void [FLEXIO\\_SPI\\_Enable](#) ([FLEXIO\\_SPI\\_Type](#) \*base, bool enable)  
*Enables/disables the FlexIO SPI module operation.*
- void [FLEXIO\\_SPI\\_MasterSetBaudRate](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [uint32\\_t](#) baudRate\_Bps, [uint32\\_t](#) srcClockHz)  
*Sets baud rate for the FlexIO SPI transfer, which is only used for the master.*
- static void [FLEXIO\\_SPI\\_WriteData](#) ([FLEXIO\\_SPI\\_Type](#) \*base, [flexio\\_spi\\_shift\\_direction\\_t](#) direction, [uint16\\_t](#) data)  
*Writes one byte of data, which is sent using the MSB method.*

## FlexIO SPI Driver

- 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.*

## 23.7.3 Data Structure Documentation

### 23.7.3.1 struct FLEXIO\_SPI\_Type

#### Data Fields

- FLEXIO\_Type \* [flexioBase](#)  
*FlexIO base pointer.*
- uint8\_t [SDOPinIndex](#)  
*Pin select for data output.*
- uint8\_t [SDIPinIndex](#)  
*Pin select for data input.*
- uint8\_t [SCKPinIndex](#)  
*Pin select for clock.*
- uint8\_t [CSnPinIndex](#)  
*Pin select for enable.*
- uint8\_t [shifterIndex](#) [2]  
*Shifter index used in FlexIO SPI.*
- uint8\_t [timerIndex](#) [2]  
*Timer index used in FlexIO SPI.*

#### 23.7.3.1.0.5 Field Documentation

23.7.3.1.0.5.1 FLEXIO\_Type\* FLEXIO\_SPI\_Type::flexioBase

23.7.3.1.0.5.2 uint8\_t FLEXIO\_SPI\_Type::SDOPinIndex

23.7.3.1.0.5.3 uint8\_t FLEXIO\_SPI\_Type::SDIPinIndex

23.7.3.1.0.5.4 uint8\_t FLEXIO\_SPI\_Type::SCKPinIndex

23.7.3.1.0.5.5 uint8\_t FLEXIO\_SPI\_Type::CSnPinIndex

23.7.3.1.0.5.6 uint8\_t FLEXIO\_SPI\_Type::shifterIndex[2]

23.7.3.1.0.5.7 uint8\_t FLEXIO\_SPI\_Type::timerIndex[2]

### 23.7.3.2 struct flexio\_spi\_master\_config\_t

#### Data Fields

- bool [enableMaster](#)  
*Enable/disable FlexIO SPI master after configuration.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*
- bool [enableFastAccess](#)  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

## FlexIO SPI Driver

- `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.*

### 23.7.3.2.0.6 Field Documentation

23.7.3.2.0.6.1 `bool flexio_spi_master_config_t::enableMaster`

23.7.3.2.0.6.2 `bool flexio_spi_master_config_t::enableInDoze`

23.7.3.2.0.6.3 `bool flexio_spi_master_config_t::enableInDebug`

23.7.3.2.0.6.4 `bool flexio_spi_master_config_t::enableFastAccess`

23.7.3.2.0.6.5 `uint32_t flexio_spi_master_config_t::baudRate_Bps`

23.7.3.2.0.6.6 `flexio_spi_clock_phase_t flexio_spi_master_config_t::phase`

23.7.3.2.0.6.7 `flexio_spi_data_bitcount_mode_t flexio_spi_master_config_t::dataMode`

### 23.7.3.3 struct flexio\_spi\_slave\_config\_t

#### Data Fields

- `bool enableSlave`  
*Enable/disable FlexIO SPI slave after configuration.*
- `bool enableInDoze`  
*Enable/disable FlexIO operation in doze mode.*
- `bool enableInDebug`  
*Enable/disable FlexIO operation in debug mode.*
- `bool enableFastAccess`  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- `flexio_spi_clock_phase_t phase`  
*Clock phase.*
- `flexio_spi_data_bitcount_mode_t dataMode`  
*8bit or 16bit mode.*

### 23.7.3.3.0.7 Field Documentation

23.7.3.3.0.7.1 `bool flexio_spi_slave_config_t::enableSlave`

23.7.3.3.0.7.2 `bool flexio_spi_slave_config_t::enableInDoze`

23.7.3.3.0.7.3 `bool flexio_spi_slave_config_t::enableInDebug`

23.7.3.3.0.7.4 `bool flexio_spi_slave_config_t::enableFastAccess`

23.7.3.3.0.7.5 `flexio_spi_clock_phase_t flexio_spi_slave_config_t::phase`

23.7.3.3.0.7.6 `flexio_spi_data_bitcount_mode_t flexio_spi_slave_config_t::dataMode`

### 23.7.3.4 struct `flexio_spi_transfer_t`

#### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t dataSize`  
*Transfer bytes.*
- `uint8_t flags`  
*FlexIO SPI control flag, MSB first or LSB first.*

### 23.7.3.4.0.8 Field Documentation

23.7.3.4.0.8.1 `uint8_t* flexio_spi_transfer_t::txData`

23.7.3.4.0.8.2 `uint8_t* flexio_spi_transfer_t::rxData`

23.7.3.4.0.8.3 `size_t flexio_spi_transfer_t::dataSize`

23.7.3.4.0.8.4 `uint8_t flexio_spi_transfer_t::flags`

### 23.7.3.5 struct `_flexio_spi_master_handle`

typedef for `flexio_spi_master_handle_t` in advance.

#### Data Fields

- `uint8_t * txData`  
*Transfer buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `volatile size_t txRemainingBytes`

## FlexIO SPI Driver

- *Send data remaining in bytes.*  
volatile size\_t **rxRemainingBytes**
- *Receive data remaining in bytes.*  
volatile uint32\_t **state**  
*FlexIO SPI internal state.*
- uint8\_t **bytePerFrame**  
*SPI mode, 2bytes or 1byte in a frame.*
- flexio\_spi\_shift\_direction\_t **direction**  
*Shift direction.*
- flexio\_spi\_master\_transfer\_callback\_t **callback**  
*FlexIO SPI callback.*
- void \* **userData**  
*Callback parameter.*



### 23.7.3.5.0.9 Field Documentation

23.7.3.5.0.9.1 `uint8_t* flexio_spi_master_handle_t::txData`

23.7.3.5.0.9.2 `uint8_t* flexio_spi_master_handle_t::rxData`

23.7.3.5.0.9.3 `size_t flexio_spi_master_handle_t::transferSize`

23.7.3.5.0.9.4 `volatile size_t flexio_spi_master_handle_t::txRemainingBytes`

23.7.3.5.0.9.5 `volatile size_t flexio_spi_master_handle_t::rxRemainingBytes`

23.7.3.5.0.9.6 `volatile uint32_t flexio_spi_master_handle_t::state`

23.7.3.5.0.9.7 `flexio_spi_shift_direction_t flexio_spi_master_handle_t::direction`

23.7.3.5.0.9.8 `flexio_spi_master_transfer_callback_t flexio_spi_master_handle_t::callback`

23.7.3.5.0.9.9 `void* flexio_spi_master_handle_t::userData`

### 23.7.4 Macro Definition Documentation

23.7.4.1 `#define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

23.7.4.2 `#define FLEXIO_SPI_DUMMYDATA (0xFFFFU)`

### 23.7.5 Typedef Documentation

23.7.5.1 `typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t`

### 23.7.6 Enumeration Type Documentation

#### 23.7.6.1 `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.

#### 23.7.6.2 `enum flexio_spi_clock_phase_t`

Enumerator

*kFLEXIO\_SPI\_ClockPhaseFirstEdge* First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.

## FlexIO SPI Driver

*kFLEXIO\_SPI\_ClockPhaseSecondEdge* First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

### 23.7.6.3 enum flexio\_spi\_shift\_direction\_t

Enumerator

*kFLEXIO\_SPI\_MsbFirst* Data transfers start with most significant bit.

*kFLEXIO\_SPI\_LsbFirst* Data transfers start with least significant bit.

### 23.7.6.4 enum flexio\_spi\_data\_bitcount\_mode\_t

Enumerator

*kFLEXIO\_SPI\_8BitMode* 8-bit data transmission mode.

*kFLEXIO\_SPI\_16BitMode* 16-bit data transmission mode.

### 23.7.6.5 enum \_flexio\_spi\_interrupt\_enable

Enumerator

*kFLEXIO\_SPI\_TxEmptyInterruptEnable* Transmit buffer empty interrupt enable.

*kFLEXIO\_SPI\_RxFullInterruptEnable* Receive buffer full interrupt enable.

### 23.7.6.6 enum \_flexio\_spi\_status\_flags

Enumerator

*kFLEXIO\_SPI\_TxBufferEmptyFlag* Transmit buffer empty flag.

*kFLEXIO\_SPI\_RxBufferFullFlag* Receive buffer full flag.

### 23.7.6.7 enum \_flexio\_spi\_dma\_enable

Enumerator

*kFLEXIO\_SPI\_TxDmaEnable* Tx DMA request source.

*kFLEXIO\_SPI\_RxDmaEnable* Rx DMA request source.

*kFLEXIO\_SPI\_DmaAllEnable* All DMA request source.

### 23.7.6.8 enum `_flexio_spi_transfer_flags`

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.

## 23.7.7 Function Documentation

### 23.7.7.1 void `FLEXIO_SPI_MasterInit ( FLEXIO_SPI_Type * base, flexio_spi_master_config_t * masterConfig, uint32_t srcClock_Hz )`

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_SPI\\_MasterGetDefaultConfig\(\)](#).

Note

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

---

## FlexIO SPI Driver

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.            |
| <i>masterConfig</i> | Pointer to the <a href="#">flexio_spi_master_config_t</a> structure. |
| <i>srcClock_Hz</i>  | FlexIO source clock in Hz.                                           |

### 23.7.7.2 void FLEXIO\_SPI\_MasterDeinit ( FLEXIO\_SPI\_Type \* *base* )

Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
|-------------|--------------------------------------------------|

### 23.7.7.3 void FLEXIO\_SPI\_MasterGetDefaultConfig ( flexio\_spi\_master\_config\_t \* *masterConfig* )

The configuration can be used directly by calling the [FLEXIO\\_SPI\\_MasterConfigure\(\)](#). Example:

```
flexio_spi_master_config_t masterConfig;  
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <i>masterConfig</i> | Pointer to the <a href="#">flexio_spi_master_config_t</a> structure. |
|---------------------|----------------------------------------------------------------------|

### 23.7.7.4 void FLEXIO\_SPI\_SlaveInit ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_slave\_config\_t \* *slaveConfig* )

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO\\_SPI\\_SlaveGetDefaultConfig\(\)](#).

Note

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,  
};
```

```

.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);

```

## Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>base</i>        | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.           |
| <i>slaveConfig</i> | Pointer to the <a href="#">flexio_spi_slave_config_t</a> structure. |

### 23.7.7.5 void FLEXIO\_SPI\_SlaveDeinit ( FLEXIO\_SPI\_Type \* *base* )

## Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
|-------------|--------------------------------------------------|

### 23.7.7.6 void FLEXIO\_SPI\_SlaveGetDefaultConfig ( flexio\_spi\_slave\_config\_t \* *slaveConfig* )

The configuration can be used directly for calling the FLEXIO\_SPI\_SlaveConfigure(). Example:

```

flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);

```

## Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>slaveConfig</i> | Pointer to the <a href="#">flexio_spi_slave_config_t</a> structure. |
|--------------------|---------------------------------------------------------------------|

### 23.7.7.7 uint32\_t FLEXIO\_SPI\_GetStatusFlags ( FLEXIO\_SPI\_Type \* *base* )

## Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
|-------------|-----------------------------------------------------------|

## Returns

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

- kFLEXIO\_SPI\_TxEmptyFlag
- kFLEXIO\_SPI\_RxEmptyFlag

## FlexIO SPI Driver

23.7.7.8 void FLEXIO\_SPI\_ClearStatusFlags ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )

## Parameters

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

### 23.7.7.9 void FLEXIO\_SPI\_EnableInterrupts ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )

This function enables the FlexIO SPI interrupt.

## Parameters

|             |                                                                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                                                                                                                                           |
| <i>mask</i> | interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_SPI_RxFullInterruptEnable</li> <li>• kFLEXIO_SPI_TxEmptyInterruptEnable</li> </ul> |

### 23.7.7.10 void FLEXIO\_SPI\_DisableInterrupts ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask* )

This function disables the FlexIO SPI interrupt.

## Parameters

|             |                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                                                                                                                                          |
| <i>mask</i> | interrupt source The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_SPI_RxFullInterruptEnable</li> <li>• kFLEXIO_SPI_TxEmptyInterruptEnable</li> </ul> |

### 23.7.7.11 void FLEXIO\_SPI\_EnableDMA ( FLEXIO\_SPI\_Type \* *base*, uint32\_t *mask*, bool *enable* )

This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO\_SPI\_TxEmptyFlag does/doesn't trigger the DMA request.

## FlexIO SPI Driver

### Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>mask</i>   | SPI DMA source.                                           |
| <i>enable</i> | True means enable DMA, false means disable DMA.           |

**23.7.7.12** `static uint32_t FLEXIO_SPI_GetTxDataRegisterAddress ( FLEXIO_SPI_Type * base, flexio_spi_shift_direction_t direction ) [inline], [static]`

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

### Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

### Returns

FlexIO SPI transmit data register address.

**23.7.7.13** `static uint32_t FLEXIO_SPI_GetRxDataRegisterAddress ( FLEXIO_SPI_Type * base, flexio_spi_shift_direction_t direction ) [inline], [static]`

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

### Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

### Returns

FlexIO SPI receive data register address.

**23.7.7.14** `static void FLEXIO_SPI_Enable ( FLEXIO_SPI_Type * base, bool enable ) [inline], [static]`



## Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> . |
| <i>enable</i> | True to enable, false does not have any effect.  |

**23.7.7.15** `void FLEXIO_SPI_MasterSetBaudRate ( FLEXIO_SPI_Type * base, uint32_t baudRate_Bps, uint32_t srcClockHz )`

## Parameters

|                     |                                                           |
|---------------------|-----------------------------------------------------------|
| <i>base</i>         | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>baudRate_Bps</i> | Baud Rate needed in Hz.                                   |
| <i>srcClockHz</i>   | SPI source clock frequency in Hz.                         |

**23.7.7.16** `static void FLEXIO_SPI_WriteData ( FLEXIO_SPI_Type * base, flexio_spi_shift_direction_t direction, 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.                                        |

**23.7.7.17** `static uint16_t FLEXIO_SPI_ReadData ( FLEXIO_SPI_Type * base, flexio_spi_shift_direction_t direction ) [inline], [static]`

## Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

## FlexIO SPI Driver

### Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>direction</i> | Shift direction of MSB first or LSB first.                |

### Returns

8 bit/16 bit data received.

**23.7.7.18 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.                         |

**23.7.7.19 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.                   |

|                  |                                            |
|------------------|--------------------------------------------|
| <i>size</i>      | The number of data bytes to be received.   |
| <i>direction</i> | Shift direction of MSB first or LSB first. |

### 23.7.7.20 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> . |

### 23.7.7.21 status\_t FLEXIO\_SPI\_MasterTransferCreateHandle ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_master\_handle\_t \* *handle*, flexio\_spi\_master\_transfer\_callback\_t *callback*, void \* *userData* )

Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                        |
| <i>handle</i>   | Pointer to the flexio_spi_master_handle_t structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                           |
| <i>userData</i> | The parameter of the callback function.                                          |

Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

### 23.7.7.22 status\_t FLEXIO\_SPI\_MasterTransferNonBlocking ( FLEXIO\_SPI\_Type \* *base*, flexio\_spi\_master\_handle\_t \* *handle*, flexio\_spi\_transfer\_t \* *xfer* )

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

## FlexIO SPI Driver

### Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | FlexIO SPI transfer structure. See <a href="#">flexio_spi_transfer_t</a> .                    |

### Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_FLEXIO_SPI_Busy</i> | SPI is not idle, is running another transfer. |

**23.7.7.23** `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. |

**23.7.7.24** `status_t FLEXIO_SPI_MasterTransferGetCount ( FLEXIO_SPI_Type * base, flexio_spi_master_handle_t * handle, size_t * count )`

### Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>handle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                           |

### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

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

## Parameters

|                  |                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------|
| <i>spiType</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                     |
| <i>spiHandle</i> | Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state. |

**23.7.7.26** `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 <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                       |
| <i>userData</i> | The parameter of the callback function.                                                      |

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

**23.7.7.27** `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 <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                    |
| <i>xfer</i>   | FlexIO SPI transfer structure. See <a href="#">flexio_spi_transfer_t</a> .                   |

## Return values

|                                |                                                  |
|--------------------------------|--------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                   |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                       |
| <i>kStatus_FLEXIO_SPI_Busy</i> | SPI is not idle; it is running another transfer. |

## FlexIO SPI Driver

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

## Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_SPI_Type</a> structure.                                    |
| <i>handle</i> | Pointer to the <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |

**23.7.7.29** `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 <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                          |

## Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

**23.7.7.30** `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 <code>flexio_spi_slave_handle_t</code> structure to store the transfer state. |

## FlexIO SPI Driver

### 23.7.8 FlexIO eDMA SPI Driver

#### 23.7.8.1 Overview

#### Data Structures

- struct `flexio_spi_master_edma_handle_t`  
*FlexIO SPI eDMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef  
`flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t`  
*Slave handle is the same with master handle.*
- typedef void(\* `flexio_spi_master_edma_transfer_callback_t`)(`FLEXIO_SPI_Type *base`, `flexio_spi_master_edma_handle_t *handle`, `status_t status`, void \*userData)  
*FlexIO SPI master callback for finished transmit.*
- typedef void(\* `flexio_spi_slave_edma_transfer_callback_t`)(`FLEXIO_SPI_Type *base`, `flexio_spi_slave_edma_handle_t *handle`, `status_t status`, void \*userData)  
*FlexIO SPI slave callback for finished transmit.*

#### Driver version

- #define `FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 3))  
*FlexIO SPI EDMA driver version 2.1.3.*

#### 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.*
- `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.*

## 23.7.8.2 Data Structure Documentation

### 23.7.8.2.1 struct flexio\_spi\_master\_edma\_handle

typedef for `flexio_spi_master_edma_handle_t` in advance.

#### Data Fields

- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `bool txInProgress`  
*Send transfer in progress.*
- `bool rxInProgress`  
*Receive transfer in progress.*
- `edma_handle_t * txHandle`  
*DMA handler for SPI send.*
- `edma_handle_t * rxHandle`  
*DMA handler for SPI receive.*
- `flexio_spi_master_edma_transfer_callback_t callback`  
*Callback for SPI DMA transfer.*
- `void * userData`  
*User Data for SPI DMA callback.*

## FlexIO SPI Driver

### 23.7.8.2.1.1 Field Documentation

23.7.8.2.1.1.1 `size_t flexio_spi_master_edma_handle_t::transferSize`

23.7.8.2.1.1.2 `uint8_t flexio_spi_master_edma_handle_t::nbytes`

### 23.7.8.3 Macro Definition Documentation

23.7.8.3.1 `#define FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

### 23.7.8.4 Typedef Documentation

23.7.8.4.1 `typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t`

### 23.7.8.5 Function Documentation

23.7.8.5.1 `status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA ( FLEXIO_SPI_Type * base, flexio_spi_master_edma_handle_t * handle, flexio_spi_master_edma_transfer_callback_t callback, void * userData, edma_handle_t * txHandle, edma_handle_t * rxHandle )`

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                          |
| <i>handle</i>   | Pointer to <code>flexio_spi_master_edma_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                          |
| <i>userData</i> | callback function parameter.                                                                   |
| <i>txHandle</i> | User requested eDMA handle for FlexIO SPI RX eDMA transfer.                                    |
| <i>rxHandle</i> | User requested eDMA handle for FlexIO SPI TX eDMA transfer.                                    |

Return values

|                           |                                                     |
|---------------------------|-----------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                     |
| <i>kStatus_OutOfRange</i> | The FlexIO SPI eDMA type/handle table out of range. |

23.7.8.5.2 `status_t FLEXIO_SPI_MasterTransferEDMA ( FLEXIO_SPI_Type * base, flexio_spi_master_edma_handle_t * handle, flexio_spi_transfer_t * xfer )`

## Note

This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_MasterGetTransferCountEDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

## Parameters

|               |                                                                                                |
|---------------|------------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                          |
| <i>handle</i> | Pointer to <code>flexio_spi_master_edma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                      |

## Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

**23.7.8.5.3** `void FLEXIO_SPI_MasterTransferAbortEDMA ( FLEXIO_SPI_Type * base, flexio_spi_master_edma_handle_t * handle )`

## Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure. |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                       |

**23.7.8.5.4** `status_t FLEXIO_SPI_MasterTransferGetCountEDMA ( FLEXIO_SPI_Type * base, flexio_spi_master_edma_handle_t * handle, size_t * count )`

## Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                                     |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## FlexIO SPI Driver

**23.7.8.5.5** `static void FLEXIO_SPI_SlaveTransferCreateHandleEDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_edma_handle_t * handle, flexio_spi_slave_edma_transfer_callback_t callback, void * userData, edma_handle_t * txHandle, edma_handle_t * rxHandle )`  
`[inline], [static]`

This function initializes the FlexIO SPI slave eDMA handle.

## Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i>   | Pointer to <code>flexio_spi_slave_edma_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | SPI callback, NULL means no callback.                                                         |
| <i>userData</i> | callback function parameter.                                                                  |
| <i>txHandle</i> | User requested eDMA handle for FlexIO SPI TX eDMA transfer.                                   |
| <i>rxHandle</i> | User requested eDMA handle for FlexIO SPI RX eDMA transfer.                                   |

### 23.7.8.5.6 `status_t FLEXIO_SPI_SlaveTransferEDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_edma_handle_t * handle, flexio_spi_transfer_t * xfer )`

## Note

This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_SlaveGetTransferCountEDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

## Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i> | Pointer to <code>flexio_spi_slave_edma_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | Pointer to FlexIO SPI transfer structure.                                                     |

## Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                           |
| <i>kStatus_FLEXIO_SPI_Busy</i> | FlexIO SPI is not idle, is running another transfer. |

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

## Parameters

---

## FlexIO SPI Driver

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

**23.7.8.5.8** `static status_t FLEXIO_SPI_SlaveTransferGetCountEDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_edma_handle_t * handle, size_t * count ) [inline], [static]`

### Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.               |
| <i>handle</i> | FlexIO SPI eDMA handle pointer.                                     |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## 23.7.9 FlexIO DMA SPI Driver

### 23.7.9.1 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.*

#### Driver version

- #define `FSL_FLEXIO_SPI_DMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 3))  
*FlexIO SPI DMA driver version 2.1.3.*

#### 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.*

## FlexIO SPI Driver

- `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.*

### 23.7.9.2 Data Structure Documentation

#### 23.7.9.2.1 `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.*



### 23.7.9.2.1.1 Field Documentation

23.7.9.2.1.1.1 `size_t flexio_spi_master_dma_handle_t::transferSize`

### 23.7.9.3 Macro Definition Documentation

23.7.9.3.1 `#define FSL_FLEXIO_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

### 23.7.9.4 Typedef Documentation

23.7.9.4.1 `typedef flexio_spi_master_dma_handle_t flexio_spi_slave_dma_handle_t`

### 23.7.9.5 Function Documentation

23.7.9.5.1 `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 FLEXIO SPI master DMA handle which can be used for other FLEXIO SPI master transactional APIs. Usually, for a specified FLEXIO SPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i>   | Pointer to <code>flexio_spi_master_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. |

23.7.9.5.2 `status_t FLEXIO_SPI_MasterTransferDMA ( FLEXIO_SPI_Type * base, flexio_spi_master_dma_handle_t * handle, flexio_spi_transfer_t * xfer )`

## FlexIO SPI Driver

### Note

This interface returned immediately after transfer initiates. Call `FLEXIO_SPI_MasterGetTransferCountDMA` to poll the transfer status to check whether the FlexIO SPI transfer is finished.

### Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                         |
| <i>handle</i> | Pointer to <code>flexio_spi_master_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. |

**23.7.9.5.3** `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.                        |

**23.7.9.5.4** `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. |

**23.7.9.5.5** `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.

## 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>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.                                    |

### 23.7.9.5.6 `status_t FLEXIO_SPI_SlaveTransferDMA ( FLEXIO_SPI_Type * base, flexio_spi_slave_dma_handle_t * handle, flexio_spi_transfer_t * xfer )`

#### Note

This interface returns immediately after transfer initiates. Call `FLEXIO_SPI_SlaveGetTransferCountDMA` to poll the transfer status and check whether the FlexIO SPI transfer is finished.

### Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_SPI_Type</a> structure.                                        |
| <i>handle</i> | Pointer to <code>flexio_spi_slave_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. |

### 23.7.9.5.7 `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 flexio_spi_slave_dma_handle_t structure to store the transfer state. |

**23.7.9.5.8** `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. |

### 23.8 FlexIO UART Driver

#### 23.8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) function using the Flexible I/O.

FlexIO UART driver includes functional APIs and transactional APIs. Functional APIs target low-level APIs. Functional APIs can be used for the FlexIO UART initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO UART peripheral and how to organize functional APIs to meet the application requirements. All functional API use the `FLEXIO_UART_Type *` as the first parameter. FlexIO UART functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_uart_handle_t` as the second parameter. Initialize the handle by calling the `FLEXIO_UART_TransferCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXIO_UART_SendNonBlocking()` and `FLEXIO_UART_ReceiveNonBlocking()` set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the `kStatus_FLEXIO_UART_TxIdle` and `kStatus_FLEXIO_UART_RxIdle` status.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size through calling the `FLEXIO_UART_InstallRingBuffer()`. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The function `FLEXIO_UART_ReceiveNonBlocking()` first gets data from the ring buffer. If ring buffer does not have enough data, the function returns the data to the ring buffer and saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `statuskStatus_FLEXIO_UART_RxIdle` status.

If the receive ring buffer is full, the upper layer is informed through a callback with status `kStatus_FLEXIO_UART_RxRingBufferOverrun`. In the callback function, the upper layer reads data from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when calling the `FLEXIO_UART_InstallRingBuffer`. Note that one byte is reserved for the ring buffer maintenance. Create a handle as follows.

```
FLEXIO_UART_InstallRingBuffer(&uartDev, &handle, &ringBuffer, 32);
```

In this example, the buffer size is 32. However, only 31 bytes are used for saving data.

#### 23.8.2 Typical use case

##### 23.8.2.1 FlexIO UART send/receive using a polling method

```
uint8_t ch;
```

```

FLEXIO_UART_Type uartDev;
status_t result = kStatus_Success;
flexio_uart_user_config user_config;
FLEXIO_UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableUart = true;

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
    return;
}
FLEXIO_UART_WriteBlocking(&uartDev, txbuff, sizeof(txbuff));

while(1)
{
    FLEXIO_UART_ReadBlocking(&uartDev, &ch, 1);
    FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
}

```

### 23.8.2.2 FlexIO UART send/receive using an interrupt method

```

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

```

## FlexIO UART Driver

```
uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 120000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
    return;
}

FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
    FLEXIO_UART_UserCallback, NULL);

// Prepares to send.
sendXfer.data = sendData;
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_UART_SendNonBlocking(&uartDev, &g_uartHandle, &sendXfer);

// Send finished.
while (!txFinished)
{
}

// Prepares to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
rxFinished = false;

// Receives.
FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, NULL);

// Receive finished.
while (!rxFinished)
{
}

// ...
}
```

### 23.8.2.3 FlexIO UART receive using the ringbuffer feature

```
#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE 32

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
{
}
```



```

    userData = userData;

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;

    result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
        FLEXIO_UART_UserCallback, NULL);
    FLEXIO_UART_InstallRingBuffer(&uartDev, &g_uartHandle, ringBuffer, RING_BUFFER_SIZE);

    // Receive is working in the background to the ring buffer.

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = RX_DATA_SIZE;
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, &bytesRead);

    if (bytesRead == RX_DATA_SIZE) /* Have read enough data. */
    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}

```

### 23.8.2.4 FlexIO UART send/receive using a DMA method

```

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
dma_handle_t g_uartTxDmaHandle;
dma_handle_t g_uartRxDmaHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;
    result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    /* Init DMAMUX. */
    DMAMUX_Init(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR)

    /* Init the DMA/EDMA module */
#ifdef FSL_FEATURE_SOC_DMA_COUNT && FSL_FEATURE_SOC_DMA_COUNT > 0U
    DMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR);
    DMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL);
    DMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#ifdef FSL_FEATURE_SOC_EDMA_COUNT && FSL_FEATURE_SOC_EDMA_COUNT > 0U
    edma_config_t edmaConfig;

    EDMA_GetDefaultConfig(&edmaConfig);
    EDMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR, &edmaConfig);
#endif
}

```

```

    EDMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
FLEXIO_UART_TX_DMA_CHANNEL);
    EDMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

    dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
shifterIndex[0]);
    dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
shifterIndex[1]);

    /* Requests DMA channels for transmit and receive. */
    DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_tx);
    DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_rx);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
FLEXIO_UART_TX_DMA_CHANNEL);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
FLEXIO_UART_RX_DMA_CHANNEL);

    FLEXIO_UART_TransferCreateHandleDMA(&uartDev, &g_uartHandle,
FLEXIO_UART_UserCallback, NULL, &g_uartTxDmaHandle, &g_uartRxDmaHandle);

    // Prepares to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_UART_SendDMA(&uartDev, &g_uartHandle, &sendXfer);

    // Send finished.
    while (!txFinished)
    {
    }

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveDMA(&uartDev, &g_uartHandle, &receiveXfer, NULL);

    // Receive finished.
    while (!rxFinished)
    {
    }

    // ...
}

```

## Modules

- [FlexIO DMA UART Driver](#)
- [FlexIO eDMA UART Driver](#)

## Data Structures

- struct [FLEXIO\\_UART\\_Type](#)

## FlexIO UART Driver

- *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, 4)`)  
*FlexIO UART driver version 2.1.4.*

## Initialization and deinitialization

- status\_t [FLEXIO\\_UART\\_Init](#) (FLEXIO\_UART\_Type \*base, const flexio\_uart\_config\_t \*userConfig, uint32\_t srcClock\_Hz)  
*Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration.*
- void [FLEXIO\\_UART\\_Deinit](#) (FLEXIO\_UART\_Type \*base)  
*Resets the FlexIO UART shifter and timer config.*
- void [FLEXIO\\_UART\\_GetDefaultConfig](#) (flexio\_uart\_config\_t \*userConfig)  
*Gets the default configuration to configure the FlexIO UART.*

## Status

- uint32\_t [FLEXIO\\_UART\\_GetStatusFlags](#) (FLEXIO\_UART\_Type \*base)  
*Gets the FlexIO UART status flags.*
- void [FLEXIO\\_UART\\_ClearStatusFlags](#) (FLEXIO\_UART\_Type \*base, uint32\_t mask)  
*Gets the FlexIO UART status flags.*

## Interrupts

- void [FLEXIO\\_UART\\_EnableInterrupts](#) (FLEXIO\_UART\_Type \*base, uint32\_t mask)  
*Enables the FlexIO UART interrupt.*
- void [FLEXIO\\_UART\\_DisableInterrupts](#) (FLEXIO\_UART\_Type \*base, uint32\_t mask)  
*Disables the FlexIO UART interrupt.*

## DMA Control

- static uint32\_t [FLEXIO\\_UART\\_GetTxDataRegisterAddress](#) (FLEXIO\_UART\_Type \*base)  
*Gets the FlexIO UART transmit data register address.*
- static uint32\_t [FLEXIO\\_UART\\_GetRxDataRegisterAddress](#) (FLEXIO\_UART\_Type \*base)  
*Gets the FlexIO UART receive data register address.*
- static void [FLEXIO\\_UART\\_EnableTxDMA](#) (FLEXIO\_UART\_Type \*base, bool enable)  
*Enables/disables the FlexIO UART transmit DMA.*
- static void [FLEXIO\\_UART\\_EnableRxDMA](#) (FLEXIO\_UART\_Type \*base, bool enable)  
*Enables/disables the FlexIO UART receive DMA.*

## Bus Operations

- static void [FLEXIO\\_UART\\_Enable](#) (FLEXIO\_UART\_Type \*base, bool enable)  
*Enables/disables the FlexIO UART module operation.*
- static void [FLEXIO\\_UART\\_WriteByte](#) (FLEXIO\_UART\_Type \*base, const uint8\_t \*buffer)  
*Writes one byte of data.*
- static void [FLEXIO\\_UART\\_ReadByte](#) (FLEXIO\_UART\_Type \*base, uint8\_t \*buffer)  
*Reads one byte of data.*

## FlexIO UART Driver

- 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 bytes sent.*
- `status_t FLEXIO_UART_TransferReceiveNonBlocking` (`FLEXIO_UART_Type *base`, `flexio_uart_handle_t *handle`, `flexio_uart_transfer_t *xfer`, `size_t *receivedBytes`)  
*Receives a buffer of data using the interrupt method.*
- void `FLEXIO_UART_TransferAbortReceive` (`FLEXIO_UART_Type *base`, `flexio_uart_handle_t *handle`)  
*Aborts the receive data which was using IRQ.*
- `status_t FLEXIO_UART_TransferGetReceiveCount` (`FLEXIO_UART_Type *base`, `flexio_uart_handle_t *handle`, `size_t *count`)  
*Gets the number of bytes received.*
- void `FLEXIO_UART_TransferHandleIRQ` (`void *uartType`, `void *uartHandle`)  
*FlexIO UART IRQ handler function.*

## 23.8.3 Data Structure Documentation

### 23.8.3.1 struct FLEXIO\_UART\_Type

#### Data Fields

- `FLEXIO_Type * flexioBase`

- *FlexIO base pointer.*  
uint8\_t [TxPinIndex](#)  
*Pin select for UART\_Tx.*
- uint8\_t [RxPinIndex](#)  
*Pin select for UART\_Rx.*
- uint8\_t [shifterIndex](#) [2]  
*Shifter index used in FlexIO UART.*
- uint8\_t [timerIndex](#) [2]  
*Timer index used in FlexIO UART.*

### 23.8.3.1.0.1 Field Documentation

23.8.3.1.0.1.1 [FLEXIO\\_Type\\* FLEXIO\\_UART\\_Type::flexioBase](#)

23.8.3.1.0.1.2 [uint8\\_t FLEXIO\\_UART\\_Type::TxPinIndex](#)

23.8.3.1.0.1.3 [uint8\\_t FLEXIO\\_UART\\_Type::RxPinIndex](#)

23.8.3.1.0.1.4 [uint8\\_t FLEXIO\\_UART\\_Type::shifterIndex\[2\]](#)

23.8.3.1.0.1.5 [uint8\\_t FLEXIO\\_UART\\_Type::timerIndex\[2\]](#)

### 23.8.3.2 struct flexio\_uart\_config\_t

#### Data Fields

- bool [enableUart](#)  
*Enable/disable FlexIO UART TX & RX.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*
- bool [enableFastAccess](#)  
*Enable/disable fast access to FlexIO registers,  
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud rate in Bps.*
- [flexio\\_uart\\_bit\\_count\\_per\\_char\\_t bitCountPerChar](#)  
*number of bits, 7/8/9 -bit*

## FlexIO UART Driver

### 23.8.3.2.0.2 Field Documentation

23.8.3.2.0.2.1 `bool flexio_uart_config_t::enableUart`

23.8.3.2.0.2.2 `bool flexio_uart_config_t::enableFastAccess`

23.8.3.2.0.2.3 `uint32_t flexio_uart_config_t::baudRate_Bps`

### 23.8.3.3 struct `flexio_uart_transfer_t`

#### Data Fields

- `uint8_t * data`  
*Transfer buffer.*
- `size_t dataSize`  
*Transfer size.*

### 23.8.3.4 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 txDataSizeAll`  
*Total bytes to be sent.*
- `size_t rxDataSizeAll`  
*Total bytes to be received.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*
- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `flexio_uart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*UART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*



### 23.8.3.4.0.3 Field Documentation

- 23.8.3.4.0.3.1 `uint8_t* volatile flexio_uart_handle_t::txData`
- 23.8.3.4.0.3.2 `volatile size_t flexio_uart_handle_t::txDataSize`
- 23.8.3.4.0.3.3 `uint8_t* volatile flexio_uart_handle_t::rxData`
- 23.8.3.4.0.3.4 `volatile size_t flexio_uart_handle_t::rxDataSize`
- 23.8.3.4.0.3.5 `size_t flexio_uart_handle_t::txDataSizeAll`
- 23.8.3.4.0.3.6 `size_t flexio_uart_handle_t::rxDataSizeAll`
- 23.8.3.4.0.3.7 `uint8_t* flexio_uart_handle_t::rxRingBuffer`
- 23.8.3.4.0.3.8 `size_t flexio_uart_handle_t::rxRingBufferSize`
- 23.8.3.4.0.3.9 `volatile uint16_t flexio_uart_handle_t::rxRingBufferHead`
- 23.8.3.4.0.3.10 `volatile uint16_t flexio_uart_handle_t::rxRingBufferTail`
- 23.8.3.4.0.3.11 `flexio_uart_transfer_callback_t flexio_uart_handle_t::callback`
- 23.8.3.4.0.3.12 `void* flexio_uart_handle_t::userData`
- 23.8.3.4.0.3.13 `volatile uint8_t flexio_uart_handle_t::txState`

### 23.8.4 Macro Definition Documentation

- 23.8.4.1 `#define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))`

### 23.8.5 Typedef Documentation

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

### 23.8.6 Enumeration Type Documentation

#### 23.8.6.1 `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.

## FlexIO UART Driver

*kStatus\_FLEXIO\_UART\_RxRingBufferOverrun* UART RX software ring buffer overrun.  
*kStatus\_FLEXIO\_UART\_RxHardwareOverrun* UART RX receiver overrun.

### 23.8.6.2 enum flexio\_uart\_bit\_count\_per\_char\_t

Enumerator

*kFLEXIO\_UART\_7BitsPerChar* 7-bit data characters  
*kFLEXIO\_UART\_8BitsPerChar* 8-bit data characters  
*kFLEXIO\_UART\_9BitsPerChar* 9-bit data characters

### 23.8.6.3 enum \_flexio\_uart\_interrupt\_enable

Enumerator

*kFLEXIO\_UART\_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.  
*kFLEXIO\_UART\_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

### 23.8.6.4 enum \_flexio\_uart\_status\_flags

Enumerator

*kFLEXIO\_UART\_TxDataRegEmptyFlag* Transmit buffer empty flag.  
*kFLEXIO\_UART\_RxDataRegFullFlag* Receive buffer full flag.  
*kFLEXIO\_UART\_RxOverRunFlag* Receive buffer over run flag.

## 23.8.7 Function Documentation

### 23.8.7.1 status\_t FLEXIO\_UART\_Init ( FLEXIO\_UART\_Type \* base, const flexio\_uart\_config\_t \* userConfig, uint32\_t srcClock\_Hz )

The configuration structure can be filled by the user or be set with default values by [FLEXIO\\_UART\\_GetDefaultConfig\(\)](#).

Example

```
FLEXIO_UART_Type base = {  
    .flexioBase = FLEXIO,  
    .TxPinIndex = 0,  
    .RxPinIndex = 1,  
    .shifterIndex = {0,1},  
    .timerIndex = {0,1}  
};  
flexio_uart_config_t config = {  
    .enableInDoze = false,  
};
```

```
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

## Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>base</i>        | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.     |
| <i>userConfig</i>  | Pointer to the <a href="#">flexio_uart_config_t</a> structure. |
| <i>srcClock_Hz</i> | FlexIO source clock in Hz.                                     |

## Return values

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kStatus_Success</i>         | Configuration success               |
| <i>kStatus_InvalidArgument</i> | Buadrate configuration out of range |

**23.8.7.2 void FLEXIO\_UART\_Deinit ( FLEXIO\_UART\_Type \* base )**

## Note

After calling this API, call the FLEXIO\_UART\_Init to use the FlexIO UART module.

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_UART_Type</a> structure |
|-------------|-------------------------------------------------------|

**23.8.7.3 void FLEXIO\_UART\_GetDefaultConfig ( flexio\_uart\_config\_t \* userConfig )**

The configuration can be used directly for calling the [FLEXIO\\_UART\\_Init\(\)](#). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

## Parameters

|                   |                                                                |
|-------------------|----------------------------------------------------------------|
| <i>userConfig</i> | Pointer to the <a href="#">flexio_uart_config_t</a> structure. |
|-------------------|----------------------------------------------------------------|

**23.8.7.4 uint32\_t FLEXIO\_UART\_GetStatusFlags ( FLEXIO\_UART\_Type \* base )**

## FlexIO UART Driver

### Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
|-------------|------------------------------------------------------------|

### Returns

FlexIO UART status flags.

### 23.8.7.5 void FLEXIO\_UART\_ClearStatusFlags ( FLEXIO\_UART\_Type \* *base*, uint32\_t *mask* )

### Parameters

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

### 23.8.7.6 void FLEXIO\_UART\_EnableInterrupts ( FLEXIO\_UART\_Type \* *base*, uint32\_t *mask* )

This function enables the FlexIO UART interrupt.

### Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>mask</i> | Interrupt source.                                          |

### 23.8.7.7 void FLEXIO\_UART\_DisableInterrupts ( FLEXIO\_UART\_Type \* *base*, uint32\_t *mask* )

This function disables the FlexIO UART interrupt.

### Parameters

---

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>mask</i> | Interrupt source.                                          |

### 23.8.7.8 **static uint32\_t FLEXIO\_UART\_GetTxDataRegisterAddress ( FLEXIO\_UART\_Type \* *base* ) [inline], [static]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
|-------------|------------------------------------------------------------|

Returns

FlexIO UART transmit data register address.

### 23.8.7.9 **static uint32\_t FLEXIO\_UART\_GetRxDataRegisterAddress ( FLEXIO\_UART\_Type \* *base* ) [inline], [static]**

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
|-------------|------------------------------------------------------------|

Returns

FlexIO UART receive data register address.

### 23.8.7.10 **static void FLEXIO\_UART\_EnableTxDMA ( FLEXIO\_UART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO\_UART\_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>enable</i> | True to enable, false to disable.                          |

---

## FlexIO UART Driver

**23.8.7.11** `static void FLEXIO_UART_EnableRxDMA ( FLEXIO_UART_Type * base, bool enable ) [inline], [static]`

This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO\_UART\_RxDataRegFullFlag does/doesn't trigger the DMA request.

## Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>enable</i> | True to enable, false to disable.                          |

**23.8.7.12** `static void FLEXIO_UART_Enable ( FLEXIO_UART_Type * base, bool enable ) [inline], [static]`

## Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> . |
| <i>enable</i> | True to enable, false does not have any effect.   |

**23.8.7.13** `static void FLEXIO_UART_WriteByte ( FLEXIO_UART_Type * base, const uint8_t * buffer ) [inline], [static]`

## Note

This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

## Parameters

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>buffer</i> | The data bytes to send.                                    |

**23.8.7.14** `static void FLEXIO_UART_ReadByte ( FLEXIO_UART_Type * base, uint8_t * buffer ) [inline], [static]`

## Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

## Parameters

## FlexIO UART Driver

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure. |
| <i>buffer</i> | The buffer to store the received bytes.                    |

### 23.8.7.15 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.                          |

### 23.8.7.16 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.                   |

### 23.8.7.17 status\_t FLEXIO\_UART\_TransferCreateHandle ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle*, flexio\_uart\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [FLEXIO\\_UART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.



## Parameters

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| <i>base</i>     | to <a href="#">FLEXIO_UART_Type</a> structure.                                          |
| <i>handle</i>   | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | The callback function.                                                                  |
| <i>userData</i> | The parameter of the callback function.                                                 |

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/ISR table out of range. |

### 23.8.7.18 void FLEXIO\_UART\_TransferStartRingBuffer ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_ReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

## Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

## Parameters

|                       |                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>           | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                                   |
| <i>handle</i>         | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.      |
| <i>ringBuffer</i>     | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | Size of the ring buffer.                                                                     |

### 23.8.7.19 void FLEXIO\_UART\_TransferStopRingBuffer ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

## FlexIO UART Driver

### Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |

### 23.8.7.20 `status_t FLEXIO_UART_TransferSendNonBlocking ( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle, flexio_uart_transfer_t * xfer )`

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the [kStatus\\_FLEXIO\\_UART\\_TxIdle](#) as status parameter.

### Note

The `kStatus_FLEXIO_UART_TxIdle` is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

### Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>xfer</i>   | FlexIO UART transfer structure. See <a href="#">flexio_uart_transfer_t</a> .            |

### Return values

|                            |                                                                                |
|----------------------------|--------------------------------------------------------------------------------|
| <i>kStatus_Success</i>     | Successfully starts the data transmission.                                     |
| <i>kStatus_UART_TxBusy</i> | Previous transmission still not finished, data not written to the TX register. |

### 23.8.7.21 `void FLEXIO_UART_TransferAbortSend ( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle )`

This function aborts the interrupt-driven data sending. Get the `remainBytes` to find out how many bytes are still not sent out.

### Parameters

---

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |

### 23.8.7.22 `status_t FLEXIO_UART_TransferGetSendCount ( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle, size_t * count )`

This function gets the number of bytes sent driven by interrupt.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | Number of bytes sent so far by the non-blocking transaction.                            |

Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |

### 23.8.7.23 `status_t FLEXIO_UART_TransferReceiveNonBlocking ( FLEXIO_UART_Type * base, flexio_uart_handle_t * handle, flexio_uart_transfer_t * xfer, size_t * receivedBytes )`

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter `kStatus_UART_RxIdle`. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to `xfer->data`. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the `xfer->data[5]`. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to `xfer->data`. When all data is received, the upper layer is notified.

## FlexIO UART Driver

### 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-<br/>_RxBusy</i> | Previous receive request is not finished.                |

### 23.8.7.24 void FLEXIO\_UART\_TransferAbortReceive ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle* )

This function aborts the receive data which was using IRQ.

### Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |

### 23.8.7.25 status\_t FLEXIO\_UART\_TransferGetReceiveCount ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the number of bytes received driven by interrupt.

### Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to the <a href="#">FLEXIO_UART_Type</a> structure.                              |
| <i>handle</i> | Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state. |
| <i>count</i>  | Number of bytes received so far by the non-blocking transaction.                        |

### Return values

|                                           |                                                   |
|-------------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferIn-<br/>Progress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>                    | Successfully return the count.                    |

**23.8.7.26 void FLEXIO\_UART\_TransferHandleIRQ ( void \* *uartType*, void \* *uartHandle* )**

This function processes the FlexIO UART transmit and receives the IRQ request.

## FlexIO UART Driver

### 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. |

## 23.8.8 FlexIO eDMA UART Driver

### 23.8.8.1 Overview

#### Data Structures

- struct `flexio_uart_edma_handle_t`  
UART eDMA handle. *More...*

#### Typedefs

- typedef void(\* `flexio_uart_edma_transfer_callback_t`)(`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `status_t` status, void \*userData)  
UART transfer callback function.

#### Driver version

- #define `FSL_FLEXIO_UART_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 4))  
FlexIO UART EDMA driver version 2.1.4.

#### eDMA transactional

- `status_t FLEXIO_UART_TransferCreateHandleEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_edma_transfer_callback_t` callback, void \*userData, `edma_handle_t` \*txEdmaHandle, `edma_handle_t` \*rxEdmaHandle)  
*Initializes the UART handle which is used in transactional functions.*
- `status_t FLEXIO_UART_TransferSendEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Sends data using eDMA.*
- `status_t FLEXIO_UART_TransferReceiveEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `flexio_uart_transfer_t` \*xfer)  
*Receives data using eDMA.*
- void `FLEXIO_UART_TransferAbortSendEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle)  
*Aborts the sent data which using eDMA.*
- void `FLEXIO_UART_TransferAbortReceiveEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle)  
*Aborts the receive data which using eDMA.*
- `status_t FLEXIO_UART_TransferGetSendCountEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes sent out.*
- `status_t FLEXIO_UART_TransferGetReceiveCountEDMA` (`FLEXIO_UART_Type` \*base, `flexio_uart_edma_handle_t` \*handle, `size_t` \*count)  
*Gets the number of bytes received.*

### 23.8.8.2 Data Structure Documentation

#### 23.8.8.2.1 struct `flexio_uart_edma_handle`

##### Data Fields

- `flexio_uart_edma_transfer_callback_t` `callback`  
*Callback function.*
- `void * userData`  
*UART callback function parameter.*
- `size_t txDataSizeAll`  
*Total bytes to be sent.*
- `size_t rxDataSizeAll`  
*Total bytes to be received.*
- `edma_handle_t * txEdmaHandle`  
*The eDMA TX channel used.*
- `edma_handle_t * rxEdmaHandle`  
*The eDMA RX channel used.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*



**23.8.8.2.1.1 Field Documentation****23.8.8.2.1.1.1 flexio\_uart\_edma\_transfer\_callback\_t flexio\_uart\_edma\_handle\_t::callback****23.8.8.2.1.1.2 void\* flexio\_uart\_edma\_handle\_t::userData****23.8.8.2.1.1.3 size\_t flexio\_uart\_edma\_handle\_t::txDataSizeAll****23.8.8.2.1.1.4 size\_t flexio\_uart\_edma\_handle\_t::rxDataSizeAll****23.8.8.2.1.1.5 edma\_handle\_t\* flexio\_uart\_edma\_handle\_t::txEdmaHandle****23.8.8.2.1.1.6 edma\_handle\_t\* flexio\_uart\_edma\_handle\_t::rxEdmaHandle****23.8.8.2.1.1.7 uint8\_t flexio\_uart\_edma\_handle\_t::nbytes****23.8.8.2.1.1.8 volatile uint8\_t flexio\_uart\_edma\_handle\_t::txState****23.8.8.3 Macro Definition Documentation****23.8.8.3.1 #define FSL\_FLEXIO\_UART\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 4))****23.8.8.4 Typedef Documentation****23.8.8.4.1 typedef void(\* flexio\_uart\_edma\_transfer\_callback\_t)(FLEXIO\_UART\_Type \*base, flexio\_uart\_edma\_handle\_t \*handle, status\_t status, void \*userData)****23.8.8.5 Function Documentation****23.8.8.5.1 status\_t FLEXIO\_UART\_TransferCreateHandleEDMA ( FLEXIO\_UART\_Type \* base, flexio\_uart\_edma\_handle\_t \* handle, flexio\_uart\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* txEdmaHandle, edma\_handle\_t \* rxEdmaHandle )**

## FlexIO UART Driver

### Parameters

|                     |                                                              |
|---------------------|--------------------------------------------------------------|
| <i>base</i>         | Pointer to <a href="#">FLEXIO_UART_Type</a> .                |
| <i>handle</i>       | Pointer to <code>flexio_uart_edma_handle_t</code> structure. |
| <i>callback</i>     | The callback function.                                       |
| <i>userData</i>     | The parameter of the callback function.                      |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.               |
| <i>txEdmaHandle</i> | User requested DMA handle for TX DMA transfer.               |

### Return values

|                           |                                                     |
|---------------------------|-----------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                     |
| <i>kStatus_OutOfRange</i> | The FlexIO SPI eDMA type/handle table out of range. |

#### 23.8.8.5.2 `status_t FLEXIO_UART_TransferSendEDMA ( FLEXIO_UART_Type * base, flexio_uart_edma_handle_t * handle, flexio_uart_transfer_t * xfer )`

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

### Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>                                |
| <i>handle</i> | UART handle pointer.                                                       |
| <i>xfer</i>   | UART eDMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> . |

### Return values

|                                   |                             |
|-----------------------------------|-----------------------------|
| <i>kStatus_Success</i>            | if succeed, others failed.  |
| <i>kStatus_FLEXIO_UART-TxBusy</i> | Previous transfer on going. |

#### 23.8.8.5.3 `status_t FLEXIO_UART_TransferReceiveEDMA ( FLEXIO_UART_Type * base, flexio_uart_edma_handle_t * handle, flexio_uart_transfer_t * xfer )`

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

## Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>                                |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure                             |
| <i>xfer</i>   | UART eDMA transfer structure, see <a href="#">flexio_uart_transfer_t</a> . |

## Return values

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

#### 23.8.8.5.4 void FLEXIO\_UART\_TransferAbortSendEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle* )

This function aborts sent data which using eDMA.

## Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>    |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure |

#### 23.8.8.5.5 void FLEXIO\_UART\_TransferAbortReceiveEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle* )

This function aborts the receive data which using eDMA.

## Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>    |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure |

#### 23.8.8.5.6 status\_t FLEXIO\_UART\_TransferGetSendCountEDMA ( FLEXIO\_UART\_Type \* *base*, flexio\_uart\_edma\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the number of bytes sent out.

## 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. |

### Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |

### 23.8.8.5.7 status\_t FLEXIO\_UART\_TransferGetReceiveCountEDMA ( FLEXIO\_UART\_Type \* base, flexio\_uart\_edma\_handle\_t \* handle, size\_t \* count )

This function gets the number of bytes received.

### Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a>                      |
| <i>handle</i> | Pointer to flexio_uart_edma_handle_t structure                   |
| <i>count</i>  | Number of bytes received so far by the non-blocking transaction. |

### Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |

## 23.8.9 FlexIO DMA UART Driver

### 23.8.9.1 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.*

#### Driver version

- #define `FSL_FLEXIO_UART_DMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 4))  
*FlexIO UART DMA driver version 2.1.4.*

#### 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 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 received.*

### 23.8.9.2 Data Structure Documentation

#### 23.8.9.2.1 struct flexio\_uart\_dma\_handle

##### Data Fields

- [flexio\\_uart\\_dma\\_transfer\\_callback\\_t](#) *callback*  
*Callback function.*
- void \* [userData](#)  
*UART callback function parameter.*
- size\_t [txDataSizeAll](#)  
*Total bytes to be sent.*
- size\_t [rxDataSizeAll](#)  
*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.*

### 23.8.9.2.1.1 Field Documentation

23.8.9.2.1.1.1 `flexio_uart_dma_transfer_callback_t flexio_uart_dma_handle_t::callback`

23.8.9.2.1.1.2 `void* flexio_uart_dma_handle_t::userData`

23.8.9.2.1.1.3 `size_t flexio_uart_dma_handle_t::txDataSizeAll`

23.8.9.2.1.1.4 `size_t flexio_uart_dma_handle_t::rxDataSizeAll`

23.8.9.2.1.1.5 `dma_handle_t* flexio_uart_dma_handle_t::txDmaHandle`

23.8.9.2.1.1.6 `dma_handle_t* flexio_uart_dma_handle_t::rxDmaHandle`

23.8.9.2.1.1.7 `volatile uint8_t flexio_uart_dma_handle_t::txState`

### 23.8.9.3 Macro Definition Documentation

23.8.9.3.1 `#define FSL_FLEXIO_UART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))`

### 23.8.9.4 Typedef Documentation

23.8.9.4.1 `typedef void(* flexio_uart_dma_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_dma_handle_t *handle, status_t status, void *userData)`

### 23.8.9.5 Function Documentation

23.8.9.5.1 `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 )`

## FlexIO UART Driver

### 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>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. |

#### 23.8.9.5.2 `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 returns right away. When all data is sent out, the send callback function is called.

### Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a> structure                            |
| <i>handle</i> | Pointer to <code>flexio_uart_dma_handle_t</code> 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. |

#### 23.8.9.5.3 `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 is received, the receive callback function is called.



## Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a> structure                            |
| <i>handle</i> | Pointer to flexio_uart_dma_handle_t 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-<br/>_RxBusy</i> | Previous transfer on going. |

#### 23.8.9.5.4 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 flexio_uart_dma_handle_t structure         |

#### 23.8.9.5.5 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 flexio_uart_dma_handle_t structure         |

#### 23.8.9.5.6 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 sent out.

## FlexIO UART Driver

### Parameters

|               |                                                              |
|---------------|--------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a> structure        |
| <i>handle</i> | Pointer to flexio_uart_dma_handle_t structure                |
| <i>count</i>  | Number of bytes sent so far by the non-blocking transaction. |

### Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |

### 23.8.9.5.7 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 received.

### Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_UART_Type</a> structure            |
| <i>handle</i> | Pointer to flexio_uart_dma_handle_t structure                    |
| <i>count</i>  | Number of bytes received so far by the non-blocking transaction. |

### Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | transfer has finished or no transfer in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                    |

# Chapter 24

## FLEXSPI: Flexible Serial Peripheral Interface Driver

### 24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flexible Serial Peripheral Interface (FLEXSPI) module of MCUXpresso SDK/i.MX devices.

FLEXSPI driver includes functional APIs and interrupt/EDMA non-blocking transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FLEXSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FLEXSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FLEXSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `flexspi_handle_t`/`flexspi_edma_handle_t` as the second parameter. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandle` API. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandleEDMA` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXSPI_TransferNonBlocking()` and `FLEXSPI_TransferEDMA()` set up data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXSPI_Idle` status.

### Modules

- [FLEXSPI eDMA Driver](#)

### Data Structures

- struct `flexspi_config_t`  
*FLEXSPI configuration structure. [More...](#)*
- struct `flexspi_device_config_t`  
*External device configuration items. [More...](#)*
- struct `flexspi_transfer_t`  
*Transfer structure for FLEXSPI. [More...](#)*
- struct `flexspi_handle_t`  
*Transfer handle structure for FLEXSPI. [More...](#)*

### Macros

- #define `FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)`  
*Formula to form FLEXSPI instructions in LUT table.*

## Overview

## Typedefs

- typedef void(\* flexspi\_transfer\_callback\_t )(FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, status\_t status, void \*userData)  
*FLEXSPI transfer callback function.*

## Enumerations

- enum \_flexspi\_status {  
kStatus\_FLEXSPI\_Busy = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 0),  
kStatus\_FLEXSPI\_SequenceExecutionTimeout = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 1),  
kStatus\_FLEXSPI\_IpCommandSequenceError = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 2),  
kStatus\_FLEXSPI\_IpCommandGrantTimeout = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 3) }  
*Status structure of FLEXSPI.*
- enum \_flexspi\_command {  
kFLEXSPI\_Command\_STOP = 0x00U,  
kFLEXSPI\_Command\_SDR = 0x01U,  
kFLEXSPI\_Command\_RADDR\_SDR = 0x02U,  
kFLEXSPI\_Command\_CADDR\_SDR = 0x03U,  
kFLEXSPI\_Command\_MODE1\_SDR = 0x04U,  
kFLEXSPI\_Command\_MODE2\_SDR = 0x05U,  
kFLEXSPI\_Command\_MODE4\_SDR = 0x06U,  
kFLEXSPI\_Command\_MODE8\_SDR = 0x07U,  
kFLEXSPI\_Command\_WRITE\_SDR = 0x08U,  
kFLEXSPI\_Command\_READ\_SDR = 0x09U,  
kFLEXSPI\_Command\_LEARN\_SDR = 0x0AU,  
kFLEXSPI\_Command\_DATSZ\_SDR = 0x0BU,  
kFLEXSPI\_Command\_DUMMY\_SDR = 0x0CU,  
kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR = 0x0DU,  
kFLEXSPI\_Command\_DDR = 0x21U,  
kFLEXSPI\_Command\_RADDR\_DDR = 0x22U,  
kFLEXSPI\_Command\_CADDR\_DDR = 0x23U,  
kFLEXSPI\_Command\_MODE1\_DDR = 0x24U,  
kFLEXSPI\_Command\_MODE2\_DDR = 0x25U,  
kFLEXSPI\_Command\_MODE4\_DDR = 0x26U,  
kFLEXSPI\_Command\_MODE8\_DDR = 0x27U,  
kFLEXSPI\_Command\_WRITE\_DDR = 0x28U,  
kFLEXSPI\_Command\_READ\_DDR = 0x29U,  
kFLEXSPI\_Command\_LEARN\_DDR = 0x2AU,  
kFLEXSPI\_Command\_DATSZ\_DDR = 0x2BU,  
kFLEXSPI\_Command\_DUMMY\_DDR = 0x2CU,  
kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR = 0x2DU,  
kFLEXSPI\_Command\_JUMP\_ON\_CS = 0x1FU }  
*CMD definition of FLEXSPI, use to form LUT instruction.*
- enum \_flexspi\_pad {

```

kFLEXSPI_1PAD = 0x00U,
kFLEXSPI_2PAD = 0x01U,
kFLEXSPI_4PAD = 0x02U,
kFLEXSPI_8PAD = 0x03U }

```

*pad definition of FLEXSPI, use to form LUT instruction.*

- enum flexspi\_flags\_t {

```

kFLEXSPI_SequenceExecutionTimeoutFlag = FLEXSPI_INTEN_SEQTIMEOUTEN_MASK,
kFLEXSPI_AhbBusTimeoutFlag = FLEXSPI_INTEN_AHBBUSTIMEOUTEN_MASK,
kFLEXSPI_SckStoppedBecauseTxEmptyFlag,
kFLEXSPI_SckStoppedBecauseRxFullFlag,
kFLEXSPI_IpTxFifoWatermarkEmptyFlag = FLEXSPI_INTEN_IPTXWEEN_MASK,
kFLEXSPI_IpRxFifoWatermarkAvailableFlag = FLEXSPI_INTEN_IPRXWAEN_MASK,
kFLEXSPI_AhbCommandSequenceErrorFlag,
kFLEXSPI_IpCommandSequenceErrorFlag = FLEXSPI_INTEN_IPCMDERREN_MASK,
kFLEXSPI_AhbCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandExcutionDoneFlag,
kFLEXSPI_AllInterruptFlags = 0xFFFU }

```

*FLEXSPI interrupt status flags.*

- enum flexspi\_read\_sample\_clock\_t {

```

kFLEXSPI_ReadSampleClkLoopbackInternally = 0x0U,
kFLEXSPI_ReadSampleClkLoopbackFromDqsPad = 0x1U,
kFLEXSPI_ReadSampleClkLoopbackFromSckPad = 0x2U,
kFLEXSPI_ReadSampleClkExternalInputFromDqsPad = 0x3U }

```

*FLEXSPI sample clock source selection for Flash Reading.*

- enum flexspi\_cs\_interval\_cycle\_unit\_t {

```

kFLEXSPI_CsIntervalUnit1SckCycle = 0x0U,
kFLEXSPI_CsIntervalUnit256SckCycle = 0x1U }

```

*FLEXSPI interval unit for flash device select.*

- enum flexspi\_ahb\_write\_wait\_unit\_t {

```

kFLEXSPI_AhbWriteWaitUnit2AhbCycle = 0x0U,
kFLEXSPI_AhbWriteWaitUnit8AhbCycle = 0x1U,
kFLEXSPI_AhbWriteWaitUnit32AhbCycle = 0x2U,
kFLEXSPI_AhbWriteWaitUnit128AhbCycle = 0x3U,
kFLEXSPI_AhbWriteWaitUnit512AhbCycle = 0x4U,
kFLEXSPI_AhbWriteWaitUnit2048AhbCycle = 0x5U,
kFLEXSPI_AhbWriteWaitUnit8192AhbCycle = 0x6U,
kFLEXSPI_AhbWriteWaitUnit32768AhbCycle = 0x7U }

```

*FLEXSPI AHB wait interval unit for writting.*

- enum flexspi\_ip\_error\_code\_t {

## Overview

```
kFLEXSPI_IpCmdErrorNoError = 0x0U,  
kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd = 0x2U,  
kFLEXSPI_IpCmdErrorUnknownOpCode = 0x3U,  
kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence = 0x4U,  
kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence = 0x5U,  
kFLEXSPI_IpCmdErrorInvalidAddress = 0x6U,  
kFLEXSPI_IpCmdErrorSequenceExecutionTimeout = 0xEU,  
kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss = 0xFU }
```

*Error Code when IP command Error detected.*

- enum flexspi\_ahb\_error\_code\_t {  
kFLEXSPI\_AhbCmdErrorNoError = 0x0U,  
kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd = 0x2U,  
kFLEXSPI\_AhbCmdErrorUnknownOpCode = 0x3U,  
kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence = 0x4U,  
kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence = 0x5U,  
kFLEXSPI\_AhbCmdSequenceExecutionTimeout = 0x6U }

*Error Code when AHB command Error detected.*

- enum flexspi\_port\_t {  
kFLEXSPI\_PortA1 = 0x0U,  
kFLEXSPI\_PortA2 = 0x1U,  
kFLEXSPI\_PortB1 = 0x2U,  
kFLEXSPI\_PortB2 = 0x3U }
- *FLEXSPI operation port select.*
- enum flexspi\_arb\_command\_source\_t  
*Trigger source of current command sequence granted by arbitrator.*
- enum flexspi\_command\_type\_t {  
kFLEXSPI\_Command,  
kFLEXSPI\_Config }

## Driver version

- #define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))  
*FLEXSPI driver version 2.0.2.*

## Initialization and deinitialization

- void FLEXSPI\_Init (FLEXSPI\_Type \*base, const flexspi\_config\_t \*config)  
*Initializes the FLEXSPI module and internal state.*
- void FLEXSPI\_GetDefaultConfig (flexspi\_config\_t \*config)  
*Gets default settings for FLEXSPI.*
- void FLEXSPI\_Deinit (FLEXSPI\_Type \*base)  
*Deinitializes the FLEXSPI module.*
- void FLEXSPI\_SetFlashConfig (FLEXSPI\_Type \*base, flexspi\_device\_config\_t \*config, flexspi\_port\_t port)  
*Configures the connected device parameter.*
- static void FLEXSPI\_SoftwareReset (FLEXSPI\_Type \*base)  
*Software reset for the FLEXSPI logic.*
- static void FLEXSPI\_Enable (FLEXSPI\_Type \*base, bool enable)

*Enables or disables the FLEXSPI module.*

## Interrupts

- static void [FLEXSPI\\_EnableInterrupts](#) (FLEXSPI\_Type \*base, uint32\_t mask)  
*Enables the FLEXSPI interrupts.*
- static void [FLEXSPI\\_DisableInterrupts](#) (FLEXSPI\_Type \*base, uint32\_t mask)  
*Disable the FLEXSPI interrupts.*

## DMA control

- static void [FLEXSPI\\_EnableTxDMA](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Tx FIFO DMA requests.*
- static void [FLEXSPI\\_EnableRxDMA](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Rx FIFO DMA requests.*
- static uint32\_t [FLEXSPI\\_GetTxFifoAddress](#) (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP tx fifo address for DMA transfer.*
- static uint32\_t [FLEXSPI\\_GetRxFifoAddress](#) (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP rx fifo address for DMA transfer.*

## FIFO control

- static void [FLEXSPI\\_ResetFifos](#) (FLEXSPI\_Type \*base, bool txFifo, bool rxFifo)  
*Clears the FLEXSPI IP FIFO logic.*
- static void [FLEXSPI\\_GetFifoCounts](#) (FLEXSPI\_Type \*base, size\_t \*txCount, size\_t \*rxCount)  
*Gets the valid data entries in the FLEXSPI FIFOs.*

## Status

- static uint32\_t [FLEXSPI\\_GetInterruptStatusFlags](#) (FLEXSPI\_Type \*base)  
*Get the FLEXSPI interrupt status flags.*
- static void [FLEXSPI\\_ClearInterruptStatusFlags](#) (FLEXSPI\_Type \*base, uint32\_t mask)  
*Get the FLEXSPI interrupt status flags.*
- static flexspi\_arb\_command\_source\_t [FLEXSPI\\_GetArbitratorCommandSource](#) (FLEXSPI\_Type \*base)  
*Gets the trigger source of current command sequence granted by arbitrator.*
- static flexspi\_ip\_error\_code\_t [FLEXSPI\\_GetIPCommandErrorCode](#) (FLEXSPI\_Type \*base, uint8\_t \*index)  
*Gets the error code when IP command error detected.*
- static flexspi\_ahb\_error\_code\_t [FLEXSPI\\_GetAHBCommandErrorCode](#) (FLEXSPI\_Type \*base, uint8\_t \*index)  
*Gets the error code when AHB command error detected.*
- static bool [FLEXSPI\\_GetBusIdleStatus](#) (FLEXSPI\_Type \*base)  
*Returns whether the bus is idle.*

## Bus Operations

- static void [FLEXSPI\\_EnableIPParallelMode](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables/disables the FLEXSPI IP command parallel mode.*
- static void [FLEXSPI\\_EnableAHBParallelMode](#) (FLEXSPI\_Type \*base, bool enable)

## Data Structure Documentation

- *Enables/disables the FLEXSPI AHB command parallel mode.*
- void [FLEXSPI\\_UpdateLUT](#) (FLEXSPI\_Type \*base, uint32\_t index, const uint32\_t \*cmd, uint32\_t count)  
*Updates the LUT table.*
- static void [FLEXSPI\\_WriteData](#) (FLEXSPI\_Type \*base, uint32\_t data, uint8\_t fifoIndex)  
*Writes data into FIFO.*
- static uint32\_t [FLEXSPI\\_ReadData](#) (FLEXSPI\_Type \*base, uint8\_t fifoIndex)  
*Receives data from data FIFO.*
- status\_t [FLEXSPI\\_WriteBlocking](#) (FLEXSPI\_Type \*base, uint32\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes using blocking method.*
- status\_t [FLEXSPI\\_ReadBlocking](#) (FLEXSPI\_Type \*base, uint32\_t \*buffer, size\_t size)  
*Receives a buffer of data bytes using a blocking method.*
- status\_t [FLEXSPI\\_TransferBlocking](#) (FLEXSPI\_Type \*base, [flexspi\\_transfer\\_t](#) \*xfer)  
*Execute command to transfer a buffer data bytes using a blocking method.*

## Transactional

- void [FLEXSPI\\_TransferCreateHandle](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle, [flexspi\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the FLEXSPI handle which is used in transactional functions.*
- status\_t [FLEXSPI\\_TransferNonBlocking](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle, [flexspi\\_transfer\\_t](#) \*xfer)  
*Performs a interrupt non-blocking transfer on the FLEXSPI bus.*
- status\_t [FLEXSPI\\_TransferGetCount](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- void [FLEXSPI\\_TransferAbort](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void [FLEXSPI\\_TransferHandleIRQ](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle)  
*Master interrupt handler.*

## 24.2 Data Structure Documentation

### 24.2.1 struct flexspi\_config\_t

#### Data Fields

- [flexspi\\_read\\_sample\\_clock\\_t rxSampleClock](#)  
*Sample Clock source selection for Flash Reading.*
- bool [enableSckFreeRunning](#)  
*Enable/disable SCK output free-running.*
- bool [enableCombination](#)  
*Enable/disable combining PORT A and B Data Pins (SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.*
- bool [enableDoze](#)  
*Enable/disable doze mode support.*
- bool [enableHalfSpeedAccess](#)  
*Enable/disable divide by 2 of the clock for half speed commands.*
- bool [enableSckBDiffOpt](#)



- Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.
- bool [enableSameConfigForAll](#)  
 Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.
- uint16\_t [seqTimeoutCycle](#)  
 Timeout wait cycle for command sequence execution, timeout after  $ahbGrantTimeoutCyle*1024$  serial root clock cycles.
- uint8\_t [ipGrantTimeoutCycle](#)  
 Timeout wait cycle for IP command grant, timeout after  $ipGrantTimeoutCycle*1024$  AHB clock cycles.
- uint8\_t [txWatermark](#)  
 FLEXSPI IP transmit watermark value.
- uint8\_t [rxWatermark](#)  
 FLEXSPI receive watermark value.
- bool [enableAHBWriteIpTxFifo](#)  
 Enable AHB bus write access to IP TX FIFO.
- bool [enableAHBWriteIpRxFifo](#)  
 Enable AHB bus write access to IP RX FIFO.
- uint8\_t [ahbGrantTimeoutCycle](#)  
 Timeout wait cycle for AHB command grant, timeout after  $ahbGrantTimeoutCyle*1024$  AHB clock cycles.
- uint16\_t [ahbBusTimeoutCycle](#)  
 Timeout wait cycle for AHB read/write access, timeout after  $ahbBusTimeoutCycle*1024$  AHB clock cycles.
- uint8\_t [resumeWaitCycle](#)  
 Wait cycle for idle state before suspended command sequence resume, timeout after  $ahbBusTimeoutCycle$  AHB clock cycles.
- flexspi\_ahbBuffer\_config\_t [buffer](#) [FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNT]  
 AHB buffer size.
- bool [enableClearAHBBufferOpt](#)  
 Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.
- bool [enableReadAddressOpt](#)  
 Enable/disable remove AHB read burst start address alignment limitation.
- bool [enableAHBPrefetch](#)  
 Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.
- bool [enableAHBBufferable](#)  
 Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command excution finished.
- bool [enableAHBCachable](#)  
 Enable AHB bus cachable read access support.

## Data Structure Documentation

### 24.2.1.0.7.1 Field Documentation

- 24.2.1.0.7.1.1 `flexspi_read_sample_clock_t flexspi_config_t::rxSampleClock`
- 24.2.1.0.7.1.2 `bool flexspi_config_t::enableSckFreeRunning`
- 24.2.1.0.7.1.3 `bool flexspi_config_t::enableCombination`
- 24.2.1.0.7.1.4 `bool flexspi_config_t::enableDoze`
- 24.2.1.0.7.1.5 `bool flexspi_config_t::enableHalfSpeedAccess`
- 24.2.1.0.7.1.6 `bool flexspi_config_t::enableSckBDiffOpt`
- 24.2.1.0.7.1.7 `bool flexspi_config_t::enableSameConfigForAll`
- 24.2.1.0.7.1.8 `uint16_t flexspi_config_t::seqTimeoutCycle`
- 24.2.1.0.7.1.9 `uint8_t flexspi_config_t::ipGrantTimeoutCycle`
- 24.2.1.0.7.1.10 `uint8_t flexspi_config_t::txWatermark`
- 24.2.1.0.7.1.11 `uint8_t flexspi_config_t::rxWatermark`
- 24.2.1.0.7.1.12 `bool flexspi_config_t::enableAHBWriteIpTxFifo`
- 24.2.1.0.7.1.13 `bool flexspi_config_t::enableAHBWriteIpRxFifo`
- 24.2.1.0.7.1.14 `uint8_t flexspi_config_t::ahbGrantTimeoutCycle`
- 24.2.1.0.7.1.15 `uint16_t flexspi_config_t::ahbBusTimeoutCycle`
- 24.2.1.0.7.1.16 `uint8_t flexspi_config_t::resumeWaitCycle`
- 24.2.1.0.7.1.17 `flexspi_ahbBuffer_config_t flexspi_config_t::buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT]`
- 24.2.1.0.7.1.18 `bool flexspi_config_t::enableClearAHBBufferOpt`
- 24.2.1.0.7.1.19 `bool flexspi_config_t::enableReadAddressOpt`

when enable, there is no AHB read burst start address alignment limitation.

24.2.1.0.7.1.20 **bool flexspi\_config\_t::enableAHBPrefetch**

24.2.1.0.7.1.21 **bool flexspi\_config\_t::enableAHBBufferable**

24.2.1.0.7.1.22 **bool flexspi\_config\_t::enableAHBCachable**

## 24.2.2 struct flexspi\_device\_config\_t

### Data Fields

- uint32\_t **flexspiRootClk**  
*FLEXSPI serial root clock.*
- bool **isSck2Enabled**  
*FLEXSPI use SCK2.*
- uint32\_t **flashSize**  
*Flash size in KByte.*
- **flexspi\_cs\_interval\_cycle\_unit\_t CSIntervalUnit**  
*CS interval unit, 1 or 256 cycle.*
- uint16\_t **CSInterval**  
*CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.*
- uint8\_t **CSHoldTime**  
*CS line hold time.*
- uint8\_t **CSSetupTime**  
*CS line setup time.*
- uint8\_t **dataValidTime**  
*Data valid time for external device.*
- uint8\_t **columnspace**  
*Column space size.*
- bool **enableWordAddress**  
*If enable word address.*
- uint8\_t **AWRSeqIndex**  
*Sequence ID for AHB write command.*
- uint8\_t **AWRSeqNumber**  
*Sequence number for AHB write command.*
- uint8\_t **ARDSeqIndex**  
*Sequence ID for AHB read command.*
- uint8\_t **ARDSeqNumber**  
*Sequence number for AHB read command.*
- **flexspi\_ahb\_write\_wait\_unit\_t AHBWriteWaitUnit**  
*AHB write wait unit.*
- uint16\_t **AHBWriteWaitInterval**  
*AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.*
- bool **enableWriteMask**  
*Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.*

## Data Structure Documentation

### 24.2.2.0.7.2 Field Documentation

- 24.2.2.0.7.2.1 `uint32_t flexspi_device_config_t::flexspiRootClk`
- 24.2.2.0.7.2.2 `bool flexspi_device_config_t::isSck2Enabled`
- 24.2.2.0.7.2.3 `uint32_t flexspi_device_config_t::flashSize`
- 24.2.2.0.7.2.4 `flexspi_cs_interval_cycle_unit_t flexspi_device_config_t::CSIntervalUnit`
- 24.2.2.0.7.2.5 `uint16_t flexspi_device_config_t::CSInterval`
- 24.2.2.0.7.2.6 `uint8_t flexspi_device_config_t::CSHoldTime`
- 24.2.2.0.7.2.7 `uint8_t flexspi_device_config_t::CSSetupTime`
- 24.2.2.0.7.2.8 `uint8_t flexspi_device_config_t::dataValidTime`
- 24.2.2.0.7.2.9 `uint8_t flexspi_device_config_t::columnSpace`
- 24.2.2.0.7.2.10 `bool flexspi_device_config_t::enableWordAddress`
- 24.2.2.0.7.2.11 `uint8_t flexspi_device_config_t::AWRSeqIndex`
- 24.2.2.0.7.2.12 `uint8_t flexspi_device_config_t::AWRSeqNumber`
- 24.2.2.0.7.2.13 `uint8_t flexspi_device_config_t::ARDSeqIndex`
- 24.2.2.0.7.2.14 `uint8_t flexspi_device_config_t::ARDSeqNumber`
- 24.2.2.0.7.2.15 `flexspi_ahb_write_wait_unit_t flexspi_device_config_t::AHBWriteWaitUnit`
- 24.2.2.0.7.2.16 `uint16_t flexspi_device_config_t::AHBWriteWaitInterval`
- 24.2.2.0.7.2.17 `bool flexspi_device_config_t::enableWriteMask`

### 24.2.3 struct flexspi\_transfer\_t

#### Data Fields

- `uint32_t deviceAddress`  
*Operation device address.*
- `flexspi_port_t port`  
*Operation port.*
- `flexspi_command_type_t cmdType`  
*Execution command type.*
- `uint8_t seqIndex`  
*Sequence ID for command.*
- `uint8_t SeqNumber`  
*Sequence number for command.*

- `uint32_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Data size in bytes.*

#### 24.2.3.0.7.3 Field Documentation

24.2.3.0.7.3.1 `uint32_t flexspi_transfer_t::deviceAddress`

24.2.3.0.7.3.2 `flexspi_port_t flexspi_transfer_t::port`

24.2.3.0.7.3.3 `flexspi_command_type_t flexspi_transfer_t::cmdType`

24.2.3.0.7.3.4 `uint8_t flexspi_transfer_t::seqIndex`

24.2.3.0.7.3.5 `uint8_t flexspi_transfer_t::SeqNumber`

24.2.3.0.7.3.6 `uint32_t* flexspi_transfer_t::data`

24.2.3.0.7.3.7 `size_t flexspi_transfer_t::dataSize`

### 24.2.4 struct `flexspi_handle`

#### Data Fields

- `uint32_t state`  
*Internal state for FLEXSPI transfer.*
- `uint32_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Remaining Data size in bytes.*
- `size_t transferTotalSize`  
*Total Data size in bytes.*
- `flexspi_transfer_callback_t completionCallback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*FLEXSPI callback function parameter.*

#### 24.2.4.0.7.4 Field Documentation

24.2.4.0.7.4.1 `uint32_t* flexspi_handle_t::data`

24.2.4.0.7.4.2 `size_t flexspi_handle_t::dataSize`

24.2.4.0.7.4.3 `size_t flexspi_handle_t::transferTotalSize`

24.2.4.0.7.4.4 `void* flexspi_handle_t::userData`

## Enumeration Type Documentation

### 24.3 Macro Definition Documentation

24.3.1 **#define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))**

24.3.2 **#define FLEXSPI\_LUT\_SEQ( cmd0, pad0, op0, cmd1, pad1, op1 )**

Value:

```
(FLEXSPI_LUT_OPERAND0 (op0) | FLEXSPI_LUT_NUM_PADS0 (pad0) | FLEXSPI_LUT_OPCODE0 (cmd0) | FLEXSPI_LUT_OPERAND1  
(op1) | \  
FLEXSPI_LUT_NUM_PADS1 (pad1) | FLEXSPI_LUT_OPCODE1 (cmd1))
```

### 24.4 Typedef Documentation

24.4.1 **typedef void(\* flexspi\_transfer\_callback\_t)(FLEXSPI\_Type \*base,  
flexspi\_handle\_t \*handle, status\_t status, void \*userData)**

### 24.5 Enumeration Type Documentation

#### 24.5.1 enum \_flexspi\_status

Enumerator

- kStatus\_FLEXSPI\_Busy* FLEXSPI is busy.
- kStatus\_FLEXSPI\_SequenceExecutionTimeout* Sequence execution timeout error occurred during FLEXSPI transfer.
- kStatus\_FLEXSPI\_IpCommandSequenceError* IP command Sequence execution timeout error occurred during FLEXSPI transfer.
- kStatus\_FLEXSPI\_IpCommandGrantTimeout* IP command grant timeout error occurred during FLEXSPI transfer.

#### 24.5.2 enum \_flexspi\_command

Enumerator

- kFLEXSPI\_Command\_STOP* Stop execution, deassert CS.
- kFLEXSPI\_Command\_SDR* Transmit Command code to Flash, using SDR mode.
- kFLEXSPI\_Command\_RADDR\_SDR* Transmit Row Address to Flash, using SDR mode.
- kFLEXSPI\_Command\_CADDR\_SDR* Transmit Column Address to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE1\_SDR* Transmit 1-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE2\_SDR* Transmit 2-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE4\_SDR* Transmit 4-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE8\_SDR* Transmit 8-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_WRITE\_SDR* Transmit Programming Data to Flash, using SDR mode.
- kFLEXSPI\_Command\_READ\_SDR* Receive Read Data from Flash, using SDR mode.

- kFLEXSPI\_Command\_LEARN\_SDR*** Receive Read Data or Preamble bit from Flash, SDR mode.
- kFLEXSPI\_Command\_DATSZ\_SDR*** Transmit Read/Program Data size (byte) to Flash, SDR mode.
- kFLEXSPI\_Command\_DUMMY\_SDR*** Leave data lines undriven by FlexSPI controller.
- kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.
- kFLEXSPI\_Command\_DDR*** Transmit Command code to Flash, using DDR mode.
- kFLEXSPI\_Command\_RADDR\_DDR*** Transmit Row Address to Flash, using DDR mode.
- kFLEXSPI\_Command\_CADDR\_DDR*** Transmit Column Address to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE1\_DDR*** Transmit 1-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE2\_DDR*** Transmit 2-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE4\_DDR*** Transmit 4-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_MODE8\_DDR*** Transmit 8-bit Mode bits to Flash, using DDR mode.
- kFLEXSPI\_Command\_WRITE\_DDR*** Transmit Programming Data to Flash, using DDR mode.
- kFLEXSPI\_Command\_READ\_DDR*** Receive Read Data from Flash, using DDR mode.
- kFLEXSPI\_Command\_LEARN\_DDR*** Receive Read Data or Preamble bit from Flash, DDR mode.
- kFLEXSPI\_Command\_DATSZ\_DDR*** Transmit Read/Program Data size (byte) to Flash, DDR mode.
- kFLEXSPI\_Command\_DUMMY\_DDR*** Leave data lines undriven by FlexSPI controller.
- kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.
- kFLEXSPI\_Command\_JUMP\_ON\_CS*** Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence.

### 24.5.3 enum flexspi\_pad

Enumerator

- kFLEXSPI\_1PAD*** Transmit command/address and transmit/receive data only through DATA0/DATA1.
- kFLEXSPI\_2PAD*** Transmit command/address and transmit/receive data only through DATA[1:0].
- kFLEXSPI\_4PAD*** Transmit command/address and transmit/receive data only through DATA[3:0].
- kFLEXSPI\_8PAD*** Transmit command/address and transmit/receive data only through DATA[7:0].

### 24.5.4 enum flexspi\_flags\_t

Enumerator

- kFLEXSPI\_SequenceExecutionTimeoutFlag*** Sequence execution timeout.

## Enumeration Type Documentation

*kFLEXSPI\_AhbBusTimeoutFlag* AHB Bus timeout.

*kFLEXSPI\_SckStoppedBecauseTxEmptyFlag* SCK is stopped during command sequence because Async TX FIFO empty.

*kFLEXSPI\_SckStoppedBecauseRxFullFlag* SCK is stopped during command sequence because Async RX FIFO full.

*kFLEXSPI\_IpTxFifoWatermarkEmptyFlag* IP TX FIFO WaterMark empty.

*kFLEXSPI\_IpRxFifoWatermarkAvailableFlag* IP RX FIFO WaterMark available.

*kFLEXSPI\_AhbCommandSequenceErrorFlag* AHB triggered Command Sequences Error.

*kFLEXSPI\_IpCommandSequenceErrorFlag* IP triggered Command Sequences Error.

*kFLEXSPI\_AhbCommandGrantTimeoutFlag* AHB triggered Command Sequences Grant Timeout.

*kFLEXSPI\_IpCommandGrantTimeoutFlag* IP triggered Command Sequences Grant Timeout.

*kFLEXSPI\_IpCommandExcutionDoneFlag* IP triggered Command Sequences Execution finished.

*kFLEXSPI\_AllInterruptFlags* All flags.

### 24.5.5 enum flexspi\_read\_sample\_clock\_t

Enumerator

*kFLEXSPI\_ReadSampleClkLoopbackInternally* Dummy Read strobe generated by FlexSPI Controller and loopback internally.

*kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad* Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

*kFLEXSPI\_ReadSampleClkLoopbackFromSckPad* SCK output clock and loopback from SCK pad.

*kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad* Flash provided Read strobe and input from DQS pad.

### 24.5.6 enum flexspi\_cs\_interval\_cycle\_unit\_t

Enumerator

*kFLEXSPI\_CsIntervalUnit1SckCycle* Chip selection interval: CSINTERVAL \* 1 serial clock cycle.

*kFLEXSPI\_CsIntervalUnit256SckCycle* Chip selection interval: CSINTERVAL \* 256 serial clock cycle.

### 24.5.7 enum flexspi\_ahb\_write\_wait\_unit\_t

Enumerator

*kFLEXSPI\_AhbWriteWaitUnit2AhbCycle* AWRWAIT unit is 2 ahb clock cycle.



*kFLEXSPI\_AhbWriteWaitUnit8AhbCycle* AWRWAIT unit is 8 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit32AhbCycle* AWRWAIT unit is 32 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit128AhbCycle* AWRWAIT unit is 128 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit512AhbCycle* AWRWAIT unit is 512 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle* AWRWAIT unit is 2048 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle* AWRWAIT unit is 8192 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle* AWRWAIT unit is 32768 ahb clock cycle.

### 24.5.8 enum flexspi\_ip\_error\_code\_t

Enumerator

*kFLEXSPI\_IpCmdErrorNoError* No error.  
*kFLEXSPI\_IpCmdErrorJumpOnCsInIpCmd* IP command with JMP\_ON\_CS instruction used.  
*kFLEXSPI\_IpCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.  
*kFLEXSPI\_IpCmdErrorSdrDummyInDdrSequence* Instruction DUMMY\_SDR/DUMMY\_RW-DS\_SDR used in DDR sequence.  
*kFLEXSPI\_IpCmdErrorDdrDummyInSdrSequence* Instruction DUMMY\_DDR/DUMMY\_RW-DS\_DDR used in SDR sequence.  
*kFLEXSPI\_IpCmdErrorInvalidAddress* Flash access start address exceed the whole flash address range (A1/A2/B1/B2).  
*kFLEXSPI\_IpCmdErrorSequenceExecutionTimeout* Sequence execution timeout.  
*kFLEXSPI\_IpCmdErrorFlashBoundaryAcrosss* Flash boundary crossed.

### 24.5.9 enum flexspi\_ahb\_error\_code\_t

Enumerator

*kFLEXSPI\_AhbCmdErrorNoError* No error.  
*kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd* AHB Write command with JMP\_ON\_CS instruction used in the sequence.  
*kFLEXSPI\_AhbCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.  
*kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence* Instruction DUMMY\_SDR/DUMMY\_R-WDS\_SDR used in DDR sequence.  
*kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence* Instruction DUMMY\_DDR/DUMMY\_R-WDS\_DDR used in SDR sequence.  
*kFLEXSPI\_AhbCmdSequenceExecutionTimeout* Sequence execution timeout.

### 24.5.10 enum flexspi\_port\_t

Enumerator

*kFLEXSPI\_PortA1* Access flash on A1 port.

## Function Documentation

*kFLEXSPI\_PortA2* Access flash on A2 port.

*kFLEXSPI\_PortB1* Access flash on B1 port.

*kFLEXSPI\_PortB2* Access flash on B2 port.

### 24.5.11 enum flexspi\_arb\_command\_source\_t

### 24.5.12 enum flexspi\_command\_type\_t

Enumerator

*kFLEXSPI\_Command* FlexSPI operation: Only command, both TX and Rx buffer are ignored.

*kFLEXSPI\_Config* FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

## 24.6 Function Documentation

### 24.6.1 void FLEXSPI\_Init ( FLEXSPI\_Type \* *base*, const flexspi\_config\_t \* *config* )

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | FLEXSPI configure structure.     |

### 24.6.2 void FLEXSPI\_GetDefaultConfig ( flexspi\_config\_t \* *config* )

Parameters

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

### 24.6.3 void FLEXSPI\_Deinit ( FLEXSPI\_Type \* *base* )

Clears the FLEXSPI state and FLEXSPI module registers.

## Parameters

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

#### 24.6.4 void FLEXSPI\_SetFlashConfig ( FLEXSPI\_Type \* *base*, flexspi\_device\_config\_t \* *config*, flexspi\_port\_t *port* )

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | Flash configuration parameters.  |
| <i>port</i>   | FLEXSPI Operation port.          |

#### 24.6.5 static void FLEXSPI\_SoftwareReset ( FLEXSPI\_Type \* *base* ) [inline], [static]

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

## Parameters

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

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

## Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                |
| <i>enable</i> | True means enable FLEXSPI, false means disable. |

#### 24.6.7 static void FLEXSPI\_EnableInterrupts ( FLEXSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

**24.6.8 static void FLEXSPI\_DisableInterrupts ( FLEXSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

**24.6.9 static void FLEXSPI\_EnableTxDMA ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                               |
| <i>enable</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |

**24.6.10 static void FLEXSPI\_EnableRxDMA ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                              |
| <i>enable</i> | Enable flag for receive DMA request. Pass true for enable, false for disable. |

**24.6.11 static uint32\_t FLEXSPI\_GetTxFifoAddress ( FLEXSPI\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Return values

|            |                  |
|------------|------------------|
| <i>The</i> | tx fifo address. |
|------------|------------------|

#### 24.6.12 static uint32\_t FLEXSPI\_GetRxFifoAddress ( FLEXSPI\_Type \* *base* ) [inline], [static]

## Parameters

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

## Return values

|            |                  |
|------------|------------------|
| <i>The</i> | rx fifo address. |
|------------|------------------|

#### 24.6.13 static void FLEXSPI\_ResetFifos ( FLEXSPI\_Type \* *base*, bool *txFifo*, bool *rxFifo* ) [inline], [static]

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>txFifo</i> | Pass true to reset TX FIFO.      |
| <i>rxFifo</i> | Pass true to reset RX FIFO.      |

#### 24.6.14 static void FLEXSPI\_GetFifoCounts ( FLEXSPI\_Type \* *base*, size\_t \* *txCount*, size\_t \* *rxCount* ) [inline], [static]

## Parameters

---

## Function Documentation

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | FLEXSPI peripheral base address.                                                                                             |
| out | <i>txCount</i> | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.  |

### 24.6.15 `static uint32_t FLEXSPI_GetInterruptStatusFlags ( FLEXSPI_Type * base ) [inline], [static]`

#### Parameters

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

#### Return values

|                  |                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------|
| <i>interrupt</i> | status flag, use status flag to AND <a href="#">flexspi_flags_t</a> could get the related status. |
|------------------|---------------------------------------------------------------------------------------------------|

### 24.6.16 `static void FLEXSPI_ClearInterruptStatusFlags ( FLEXSPI_Type * base, uint32_t mask ) [inline], [static]`

#### Parameters

|                  |                                  |
|------------------|----------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address. |
| <i>interrupt</i> | status flag.                     |

### 24.6.17 `static flexspi_arb_command_source_t FLEXSPI_GetArbitrator-CommandSource ( FLEXSPI_Type * base ) [inline], [static]`

#### Parameters

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

Return values

|                |                                     |
|----------------|-------------------------------------|
| <i>trigger</i> | source of current command sequence. |
|----------------|-------------------------------------|

**24.6.18** `static flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode ( FLEXSPI_Type * base, uint8_t * index ) [inline], [static]`

Parameters

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                      |
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

|              |                                      |
|--------------|--------------------------------------|
| <i>error</i> | code when IP command error detected. |
|--------------|--------------------------------------|

**24.6.19** `static flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode ( FLEXSPI_Type * base, uint8_t * index ) [inline], [static]`

Parameters

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                      |
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

|              |                                       |
|--------------|---------------------------------------|
| <i>error</i> | code when AHB command error detected. |
|--------------|---------------------------------------|

**24.6.20** `static bool FLEXSPI_GetBusIdleStatus ( FLEXSPI_Type * base ) [inline], [static]`

Parameters

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

## Function Documentation

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is idle. |
| <i>false</i> | Bus is busy. |

**24.6.21 static void FLEXSPI\_EnableIPParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

**24.6.22 static void FLEXSPI\_EnableAHBParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

**24.6.23 void FLEXSPI\_UpdateLUT ( FLEXSPI\_Type \* *base*, uint32\_t *index*, const uint32\_t \* *cmd*, uint32\_t *count* )**

Parameters

|              |                                                                                                                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                                                                                                                                                       |
| <i>index</i> | From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory. |
| <i>cmd</i>   | Command sequence array.                                                                                                                                                                                                |
| <i>count</i> | Number of sequences.                                                                                                                                                                                                   |



**24.6.24** `static void FLEXSPI_WriteData ( FLEXSPI_Type * base, uint32_t data,  
uint8_t fifoIndex ) [inline], [static]`

## Function Documentation

### Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address |
| <i>data</i>      | The data bytes to send          |
| <i>fifoIndex</i> | Destination fifo index.         |

**24.6.25** `static uint32_t FLEXSPI_ReadData ( FLEXSPI_Type * base, uint8_t fifoIndex ) [inline], [static]`

### Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address |
| <i>fifoIndex</i> | Source fifo index.              |

### Returns

The data in the FIFO.

**24.6.26** `status_t FLEXSPI_WriteBlocking ( FLEXSPI_Type * base, uint32_t * buffer, size_t size )`

## Note

This function blocks via polling until all bytes have been sent.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address  |
| <i>buffer</i> | The data bytes to send           |
| <i>size</i>   | The number of data bytes to send |

## Return values

|                                                 |                                     |
|-------------------------------------------------|-------------------------------------|
| <i>kStatus_Success</i>                          | write success without error         |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout          |
| <i>kStatus_FLEXSPI_IpCommandSequenceError</i>   | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected   |

**24.6.27** `status_t FLEXSPI_ReadBlocking ( FLEXSPI_Type * base, uint32_t * buffer, size_t size )`

## Function Documentation

### Note

This function blocks via polling until all bytes have been sent.

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address     |
| <i>buffer</i> | The data bytes to send              |
| <i>size</i>   | The number of data bytes to receive |

### Return values

|                                                 |                                     |
|-------------------------------------------------|-------------------------------------|
| <i>kStatus_Success</i>                          | read success without error          |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout          |
| <i>kStatus_FLEXSPI_IpCommandSequenceError</i>   | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected   |

### 24.6.28 **status\_t FLEXSPI\_TransferBlocking ( FLEXSPI\_Type \* *base*, flexspi\_transfer\_t \* *xfer* )**

### Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | FLEXSPI peripheral base address    |
| <i>xfer</i> | pointer to the transfer structure. |

### Return values

|                                                 |                                        |
|-------------------------------------------------|----------------------------------------|
| <i>kStatus_Success</i>                          | command transfer success without error |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout             |

|                                                |                                     |
|------------------------------------------------|-------------------------------------|
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i> | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>  | IP command grant timeout detected   |

**24.6.29 void FLEXSPI\_TransferCreateHandle ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle*, flexspi\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>base</i>     | FLEXSPI peripheral base address.                                   |
| <i>handle</i>   | pointer to flexspi_handle_t structure to store the transfer state. |
| <i>callback</i> | pointer to user callback function.                                 |
| <i>userData</i> | user parameter passed to the callback function.                    |

**24.6.30 status\_t FLEXSPI\_TransferNonBlocking ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle*, flexspi\_transfer\_t \* *xfer* )**

Note

Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI\_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus\_FLEXSPI\_Busy, the transfer is finished. For FLEXSPI\_Read, the dataSize should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                       |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | pointer to <a href="#">flexspi_transfer_t</a> structure.               |

Return values

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <i>kStatus_Success</i>      | Successfully start the data transmission. |
| <i>kStatus_FLEXSPI_Busy</i> | Previous transmission still not finished. |

---

## Function Documentation

**24.6.31** `status_t FLEXSPI_TransferGetCount ( FLEXSPI_Type * base,  
flexspi_handle_t * handle, size_t * count )`

## Parameters

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

## Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

### 24.6.32 void FLEXSPI\_TransferAbort ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle* )

## Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

## Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                      |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state |

### 24.6.33 void FLEXSPI\_TransferHandleIRQ ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle* )

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.       |
| <i>handle</i> | pointer to flexspi_handle_t structure. |

## FLEXSPI eDMA Driver

### 24.7 FLEXSPI eDMA Driver

#### 24.7.1 Overview

#### Data Structures

- struct `flexspi_edma_handle_t`  
*FLEXSPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* `flexspi_edma_callback_t`)(FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*FLEXSPI eDMA transfer callback function for finish and error.*

#### Enumerations

- enum `flexspi_edma_transfer_nsize_t` {  
    `kFLEXPSI_EDMAAnSize1Bytes` = 0x1U,  
    `kFLEXPSI_EDMAAnSize2Bytes` = 0x2U,  
    `kFLEXPSI_EDMAAnSize4Bytes` = 0x4U,  
    `kFLEXPSI_EDMAAnSize8Bytes` = 0x8U,  
    `kFLEXPSI_EDMAAnSize32Bytes` = 0x20U }  
*eDMA transfer configuration*

#### Driver version

- #define `FSL_FLEXSPI_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*FLEXSPI EDMA driver version 2.0.2.*

#### FLEXSPI eDMA Transactional

- void `FLEXSPI_TransferCreateHandleEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, flexspi\_edma\_callback\_t callback, void \*userData, edma\_handle\_t \*txDmaHandle, edma\_handle\_t \*rxDmaHandle)  
*Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.*
- void `FLEXSPI_TransferUpdateSizeEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, flexspi\_edma\_transfer\_nsize\_t nsize)  
*Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).*
- status\_t `FLEXSPI_TransferEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, flexspi\_transfer\_t \*xfer)  
*Transfers FLEXSPI data using an eDMA non-blocking method.*



- void [FLEXSPI\\_TransferAbortEDMA](#) (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle)  
*Aborts the transfer data using eDMA.*
- status\_t [FLEXSPI\\_TransferGetCountEDMA](#) (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the transferred counts of transfer.*

## 24.7.2 Data Structure Documentation

### 24.7.2.1 struct flexspi\_edma\_handle

#### Data Fields

- [edma\\_handle\\_t \\* txDmaHandle](#)  
*eDMA handler for FLEXSPI Tx.*
- [edma\\_handle\\_t \\* rxDmaHandle](#)  
*eDMA handler for FLEXSPI Rx.*
- size\_t [transferSize](#)  
*Bytes need to transfer.*
- [flexspi\\_edma\\_transfer\\_nsize\\_t nsize](#)  
*eDMA SSIZE/DSIZE in each transfer.*
- uint8\_t [nbytes](#)  
*eDMA minor byte transfer count initially configured.*
- uint8\_t [count](#)  
*The transfer data count in a DMA request.*
- uint32\_t [state](#)  
*Internal state for FLEXSPI eDMA transfer.*
- [flexspi\\_edma\\_callback\\_t completionCallback](#)  
*A callback function called after the eDMA transfer is finished.*
- void \* [userData](#)  
*User callback parameter.*

## FLEXSPI eDMA Driver

### 24.7.2.1.0.5 Field Documentation

24.7.2.1.0.5.1 `edma_handle_t* flexspi_edma_handle_t::txDmaHandle`

24.7.2.1.0.5.2 `edma_handle_t* flexspi_edma_handle_t::rxDmaHandle`

24.7.2.1.0.5.3 `size_t flexspi_edma_handle_t::transferSize`

24.7.2.1.0.5.4 `flexspi_edma_transfer_nsize_t flexspi_edma_handle_t::nsize`

24.7.2.1.0.5.5 `uint8_t flexspi_edma_handle_t::nbytes`

24.7.2.1.0.5.6 `uint8_t flexspi_edma_handle_t::count`

24.7.2.1.0.5.7 `uint32_t flexspi_edma_handle_t::state`

24.7.2.1.0.5.8 `flexspi_edma_callback_t flexspi_edma_handle_t::completionCallback`

### 24.7.3 Macro Definition Documentation

24.7.3.1 `#define FSL_FLEXSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

### 24.7.4 Enumeration Type Documentation

24.7.4.1 `enum flexspi_edma_transfer_nsize_t`

Enumerator

*kFLEXPSI\_EDMAnSize1Bytes* Source/Destination data transfer size is 1 byte every time.  
*kFLEXPSI\_EDMAnSize2Bytes* Source/Destination data transfer size is 2 bytes every time.  
*kFLEXPSI\_EDMAnSize4Bytes* Source/Destination data transfer size is 4 bytes every time.  
*kFLEXPSI\_EDMAnSize8Bytes* Source/Destination data transfer size is 8 bytes every time.  
*kFLEXPSI\_EDMAnSize32Bytes* Source/Destination data transfer size is 32 bytes every time.

### 24.7.5 Function Documentation

24.7.5.1 `void FLEXSPI_TransferCreateHandleEDMA ( FLEXSPI_Type * base,  
flexspi_edma_handle_t * handle, flexspi_edma_callback_t callback, void *  
userData, edma_handle_t * txDmaHandle, edma_handle_t * rxDmaHandle )`

Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address                |
| <i>handle</i>      | Pointer to flexspi_edma_handle_t structure     |
| <i>callback</i>    | FLEXSPI callback, NULL means no callback.      |
| <i>userData</i>    | User callback function data.                   |
| <i>txDmaHandle</i> | User requested DMA handle for TX DMA transfer. |
| <i>rxDmaHandle</i> | User requested DMA handle for RX DMA transfer. |

**24.7.5.2 void FLEXSPI\_TransferUpdateSizeEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle*, flexspi\_edma\_transfer\_nsize\_t *nsize* )**

Parameters

|               |                                                                                                                   |
|---------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address                                                                                   |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure                                                                        |
| <i>nsize</i>  | FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_EDMAAnSize1Bytes(one byte). |

See Also

[flexspi\\_edma\\_transfer\\_nsize\\_t](#) .

**24.7.5.3 status\_t FLEXSPI\_TransferEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle*, flexspi\_transfer\_t \* *xfer* )**

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.           |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure |
| <i>xfer</i>   | FLEXSPI transfer structure.                |

Return values

## FLEXSPI eDMA Driver

|                                |                                                                                                              |
|--------------------------------|--------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FLEXSPI_Busy</i>    | FLEXSPI is busy transfer.                                                                                    |
| <i>kStatus_InvalidArgument</i> | The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer. |
| <i>kStatus_Success</i>         | FLEXSPI successfully start edma transfer.                                                                    |

### 24.7.5.4 void FLEXSPI\_TransferAbortEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle* )

This function aborts the transfer data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.           |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure |

### 24.7.5.5 status\_t FLEXSPI\_TransferGetCountEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.            |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure. |
| <i>count</i>  | Bytes transfer.                             |

Return values

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

## Chapter 25

# GPC: General Power Controller Driver

### 25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Power Controller (GPC) module of MCUXpresso SDK devices.

API functions are provided to configure the system about working in dedicated power mode. There are mainly about enabling the power for memory, enabling the wakeup sources for Partial Sleep mode, and power up/down operations.

### 25.2 Typical use case

1. Enable the wakeup source for Partial Sleep (PSLEEP) mode. In PSLEEP mode, HP domain is powered down, while LP domain remains powered on. Therefore, peripherals in LP domain can wakeup the system from PSLEEP mode via interrupts. In PSLEEP mode, system clocks are stopped and peripheral clocks of LP domain can be optionally on. LP domain peripherals can generate interrupt either asynchronously or need its peripheral clock on, depending on what kind of wakeup event is expected. Using the API of "GPC\_EnablePartialSleepWakeupSource()" can enable the corresponding module as a wakeup source of PSLEEP mode. When the MCU exits from the PSLEEP mode, the API of "GPC\_GetPartialSleepWakeupFlag()" can be used to check if the expected event occurs. However, to clear the wakeup flags, the user has to clear the interrupt status of the corresponding wakeup module.
2. Power up/down sequence. After the power up/down request, the power of HP domain would not be ready immediately, it would wait for a few delay so that the voltage is stable. Then switch the power of HP domain by hardware. To set the delay time, the driver provides the API of "GPC\_ConfigPowerUpSequence()" and "GPC\_ConfigPowerDownSequence()".

### Enumerations

- enum `_gpc_memory_power_gate` {  
    `kGPC_MemoryPowerGateL2Cache` = GPC\_MPCTR\_L2\_PGE\_MASK,  
    `kGPC_MemoryPowerGateITCM` = GPC\_MPCTR\_ITCM\_PGE\_MASK,  
    `kGPC_MemoryPowerGateDTCM` = GPC\_MPCTR\_DTCM\_PGE\_MASK }  
*Enumeration of the memory power gate control.*
- enum `_gpc_status_flags` { `kGPC_PoweredDownFlag` = GPC\_PGS\_PS\_MASK }  
*GPC flags.*

### Driver version

- #define `FSL_GPC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*GPC driver version 2.0.0.*

## Enumeration Type Documentation

### Memory power Control

- static void [GPC\\_EnableMemoryGate](#) (GPC\_Type \*base, uint32\_t modules, bool enable)  
*Control power for memory.*

### Partial sleep source

- void [GPC\\_EnablePartialSleepWakeupSource](#) (GPC\_Type \*base, gpc\_wakeup\_source\_t source, bool enable)  
*Enable the modules as wakeup sources of PSLEEP (Partial Sleep) mode.*
- bool [GPC\\_GetPartialSleepWakeupFlag](#) (GPC\_Type \*base, gpc\_wakeup\_source\_t source)  
*Get if indicated wakeup module just caused the wakeup event to exit PSLEEP mode.*
- static void [GPC\\_EnablePartialSleepMode](#) (GPC\_Type \*base, bool enable)  
*Switch to the Partial Sleep mode.*

### Power gate control

- static void [GPC\\_ConfigPowerUpSequence](#) (GPC\_Type \*base, uint32\_t sw, uint32\_t sw2iso)  
*Configure the power up sequence.*
- static void [GPC\\_ConfigPowerDownSequence](#) (GPC\_Type \*base, uint32\_t iso, uint32\_t iso2w)  
*Configure the power down sequence.*
- static uint32\_t [GPC\\_GetStatusFlags](#) (GPC\_Type \*base)  
*Get the status flags of GPC.*
- static void [GPC\\_ClearStatusFlags](#) (GPC\_Type \*base, uint32\_t flags)  
*Clear the status flags of GPC.*

## 25.3 Macro Definition Documentation

### 25.3.1 #define FSL\_GPC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 25.4 Enumeration Type Documentation

### 25.4.1 enum \_gpc\_memory\_power\_gate

Once the clock gate is enabled, the responding part would be powered off and contents are not retained in Partial SLEEP mode.

Enumerator

***kGPC\_MemoryPowerGateL2Cache*** L2 Cache Power Gate.  
***kGPC\_MemoryPowerGateITCM*** ITCM Power Gate Enable.  
***kGPC\_MemoryPowerGateDTCM*** DTCM Power Gate Enable.

### 25.4.2 enum \_gpc\_status\_flags

Enumerator

***kGPC\_PoweredDownFlag*** Power status. HP domain was powered down for the previous power down request.

## 25.5 Function Documentation

25.5.1 `static void GPC_EnableMemoryGate ( GPC_Type * base, uint32_t modules, bool enable ) [inline], [static]`

## Function Documentation

### Parameters

|                |                                                                                        |
|----------------|----------------------------------------------------------------------------------------|
| <i>base</i>    | GPC peripheral base address.                                                           |
| <i>modules</i> | Mask value for Modules to be operated, see to <a href="#">_gpc_memory_power_gate</a> . |
| <i>enable</i>  | Enable the power or not.                                                               |

### 25.5.2 void GPC\_EnablePartialSleepWakeupSource ( GPC\_Type \* *base*, gpc\_wakeup\_source\_t *source*, bool *enable* )

In PSLEEP mode, HP domain is powered down, while LP domain remains powered on so peripherals in LP domain can wakeup the system from PSLEEP mode via interrupts. In PSLEEP mode, system clocks are stopped and peripheral clocks of LP domain can be optionally on. LP domain peripherals can generate interrupt either asynchronously or need its peripheral clock on, depending on what kind of wakeup event is expected. Refer to the corresponding module description about what kind of interrupts are supported by the module.

### Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | GPC peripheral base address.                                                       |
| <i>source</i> | Wakeup source for responding module, see to <a href="#">#gpc_wakeup_source_t</a> . |
| <i>enable</i> | Enable the wakeup source or not.                                                   |

### 25.5.3 bool GPC\_GetPartialSleepWakeupFlag ( GPC\_Type \* *base*, gpc\_wakeup\_source\_t *source* )

This function returns if the responding wakeup module just caused the MCU to exit PSLEEP mode. In hardware level, the flags of wakeup source are read only and will be cleared by cleaning the interrupt status of the corresponding wakeup module.

### Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | GPC peripheral base address.                                                       |
| <i>source</i> | Wakeup source for responding module, see to <a href="#">#gpc_wakeup_source_t</a> . |

### Return values

---



|              |                                          |
|--------------|------------------------------------------|
| <i>true</i>  | - Indicated wakeup flag is asserted.     |
| <i>false</i> | - Indicated wakeup flag is not asserted. |

#### 25.5.4 static void GPC\_EnablePartialSleepMode ( GPC\_Type \* *base*, bool *enable* ) [inline], [static]

This function controls if the system will enter Partial SLEEP mode or remain in STOP mode.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | GPC peripheral base address. |
| <i>enable</i> | Enable the gate or not.      |

#### 25.5.5 static void GPC\_ConfigPowerUpSequence ( GPC\_Type \* *base*, uint32\_t *sw*, uint32\_t *sw2iso* ) [inline], [static]

There will be two steps for power up sequence:

- After a power up request, GPC waits for a number of IP BUS clocks equal to the value of SW before turning on the power of HP domain. SW must not be programmed to zero.
- After GPC turning on the power of HP domain, it waits for a number of IP BUS clocks equal to the value of SW2ISO before disable the isolation of HP domain. SW2ISO must not be programmed to zero.

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | GPC peripheral base address.                                        |
| <i>sw</i>     | Count of IP BUS clocks before disabling the isolation of HP domain. |
| <i>sw2iso</i> | Count of IP BUS clocks before turning on the power of HP domain.    |

#### 25.5.6 static void GPC\_ConfigPowerDownSequence ( GPC\_Type \* *base*, uint32\_t *iso*, uint32\_t *iso2sw* ) [inline], [static]

There will be two steps for power down sequence:

- After a power down request, the GPC waits for a number of IP BUS clocks equal to the value of ISO before it enables the isolation of HP domain. ISO must not be programmed to zero.
- After HP domain is isolated, GPC waits for a number of IPG BUS clocks equal to the value of ISO2SW before it turning off the power of HP domain. ISO2SW must not be programmed to zero.

## Function Documentation

Parameters

|              |                                                                      |
|--------------|----------------------------------------------------------------------|
| <i>base</i>  | GPC peripheral base address.                                         |
| <i>iso</i>   | Count of IP BUS clocks before it enables the isolation of HP domain. |
| <i>iso2w</i> | Count of IP BUS clocks before it turning off the power of HP domain. |

**25.5.7** `static uint32_t GPC_GetStatusFlags ( GPC_Type * base ) [inline], [static]`

Parameters

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

Returns

Mask value of flags, see to [\\_gpc\\_status\\_flags](#).

**25.5.8** `static void GPC_ClearStatusFlags ( GPC_Type * base, uint32_t flags ) [inline], [static]`

Parameters

|              |                                                                               |
|--------------|-------------------------------------------------------------------------------|
| <i>base</i>  | GPC peripheral base address.                                                  |
| <i>flags</i> | Mask value of flags to be cleared, see to <a href="#">_gpc_status_flags</a> . |



## Chapter 26

# GPIO: General-Purpose Input/Output Driver

### 26.1 Overview

#### Modules

- [GPIO Driver](#)

## GPIO Driver

### 26.2 GPIO Driver

#### 26.2.1 Overview

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

#### 26.2.2 Typical use case

##### 26.2.2.1 Input Operation

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

### Data Structures

- struct `gpio_pin_config_t`  
*GPIO Init structure definition. [More...](#)*

### Enumerations

- enum `gpio_pin_direction_t` {  
    `kGPIO_DigitalInput` = 0U,  
    `kGPIO_DigitalOutput` = 1U }  
*GPIO direction definition.*
- enum `gpio_interrupt_mode_t` {  
    `kGPIO_NoIntmode` = 0U,  
    `kGPIO_IntLowLevel` = 1U,  
    `kGPIO_IntHighLevel` = 2U,  
    `kGPIO_IntRisingEdge` = 3U,  
    `kGPIO_IntFallingEdge` = 4U,  
    `kGPIO_IntRisingOrFallingEdge` = 5U }  
*GPIO interrupt mode definition.*

### Driver version

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

### GPIO Initialization and Configuration functions

- void `GPIO_PinInit` (GPIO\_Type \*base, uint32\_t pin, const `gpio_pin_config_t` \*Config)  
*Initializes the GPIO peripheral according to the specified parameters in the initConfig.*

## GPIO Reads and Write Functions

- void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void [GPIO\\_WritePinOutput](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_SetPinsOutput](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_PortClear](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_ClearPinsOutput](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_PortToggle](#) (GPIO\_Type \*base, uint32\_t mask)  
*Reverses the current output logic of the multiple GPIO pins.*
- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*
- static uint32\_t [GPIO\\_ReadPinInput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*

## GPIO Reads Pad Status Functions

- static uint8\_t [GPIO\\_PinReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*
- static uint8\_t [GPIO\\_ReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*

## Interrupts and flags management functions

- void [GPIO\\_PinSetInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_SetPinInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_PortEnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_EnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_PortDisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static void [GPIO\\_DisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static uint32\_t [GPIO\\_PortGetInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*
- static uint32\_t [GPIO\\_GetPinsInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*

## GPIO Driver

- static void [GPIO\\_PortClearInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*
- static void [GPIO\\_ClearPinsInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*

### 26.2.3 Data Structure Documentation

#### 26.2.3.1 struct gpio\_pin\_config\_t

##### Data Fields

- [gpio\\_pin\\_direction\\_t direction](#)  
*Specifies the pin direction.*
- uint8\_t [outputLogic](#)  
*Set a default output logic, which has no use in input.*
- [gpio\\_interrupt\\_mode\\_t interruptMode](#)  
*Specifies the pin interrupt mode, a value of [gpio\\_interrupt\\_mode\\_t](#).*

##### 26.2.3.1.0.6 Field Documentation

###### 26.2.3.1.0.6.1 gpio\_pin\_direction\_t gpio\_pin\_config\_t::direction

###### 26.2.3.1.0.6.2 gpio\_interrupt\_mode\_t gpio\_pin\_config\_t::interruptMode

### 26.2.4 Macro Definition Documentation

#### 26.2.4.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 26.2.5 Enumeration Type Documentation

#### 26.2.5.1 enum gpio\_pin\_direction\_t

Enumerator

- kGPIO\_DigitalInput* Set current pin as digital input.
- kGPIO\_DigitalOutput* Set current pin as digital output.

#### 26.2.5.2 enum gpio\_interrupt\_mode\_t

Enumerator

- kGPIO\_NoIntmode* Set current pin general IO functionality.
- kGPIO\_IntLowLevel* Set current pin interrupt is low-level sensitive.
- kGPIO\_IntHighLevel* Set current pin interrupt is high-level sensitive.
- kGPIO\_IntRisingEdge* Set current pin interrupt is rising-edge sensitive.

***kGPIO\_IntFallingEdge*** Set current pin interrupt is falling-edge sensitive.

***kGPIO\_IntRisingOrFallingEdge*** Enable the edge select bit to override the ICR register's configuration.

## 26.2.6 Function Documentation

### 26.2.6.1 void GPIO\_PinInit ( GPIO\_Type \* *base*, uint32\_t *pin*, const gpio\_pin\_config\_t \* *Config* )

Parameters

|                   |                                                                                                       |
|-------------------|-------------------------------------------------------------------------------------------------------|
| <i>base</i>       | GPIO base pointer.                                                                                    |
| <i>pin</i>        | Specifies the pin number                                                                              |
| <i>initConfig</i> | pointer to a <a href="#">gpio_pin_config_t</a> structure that contains the configuration information. |

### 26.2.6.2 void GPIO\_PinWrite ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* )

Parameters

|               |                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | GPIO base pointer.                                                                                                                                                                    |
| <i>pin</i>    | GPIO port pin number.                                                                                                                                                                 |
| <i>output</i> | GPIOpin 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> |

### 26.2.6.3 static void GPIO\_WritePinOutput ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

### 26.2.6.4 static void GPIO\_PortSet ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

## GPIO Driver

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

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

**26.2.6.6** `static void GPIO_PortClear ( GPIO_Type * base, uint32_t mask )` `[inline],`  
`[static]`

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

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

**26.2.6.8** `static void GPIO_PortToggle ( GPIO_Type * base, uint32_t mask )` `[inline],`  
`[static]`

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**26.2.6.9** `static uint32_t GPIO_PinRead ( GPIO_Type * base, uint32_t pin )` `[inline],`  
`[static]`

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | GPIO base pointer.    |
| <i>pin</i>  | GPIO port pin number. |

Return values

---



|             |                   |
|-------------|-------------------|
| <i>GPIO</i> | port input value. |
|-------------|-------------------|

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

**26.2.6.11** `static uint8_t GPIO_PinReadPadStatus ( GPIO_Type * base, uint32_t pin )`  
`[inline], [static]`

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | GPIO base pointer.    |
| <i>pin</i>  | GPIO port pin number. |

Return values

|             |                       |
|-------------|-----------------------|
| <i>GPIO</i> | pin pad status value. |
|-------------|-----------------------|

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

**26.2.6.13** `void GPIO_PinSetInterruptConfig ( GPIO_Type * base, uint32_t pin,`  
`gpio_interrupt_mode_t pinInterruptMode )`

Parameters

|                          |                                                                                                            |
|--------------------------|------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | GPIO base pointer.                                                                                         |
| <i>pin</i>               | GPIO port pin number.                                                                                      |
| <i>pininterrupt-Mode</i> | pointer to a <a href="#">gpio_interrupt_mode_t</a> structure that contains the interrupt mode information. |

**26.2.6.14** `static void GPIO_SetPinInterruptConfig ( GPIO_Type * base, uint32_t pin,`  
`gpio_interrupt_mode_t pinInterruptMode ) [inline], [static]`

**26.2.6.15** `static void GPIO_PortEnableInterrupts ( GPIO_Type * base, uint32_t mask )`  
`[inline], [static]`

## GPIO Driver

### Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**26.2.6.16** `static void GPIO_EnableInterrupts ( GPIO_Type * base, uint32_t mask )`  
`[inline], [static]`

### Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**26.2.6.17** `static void GPIO_PortDisableInterrupts ( GPIO_Type * base, uint32_t mask )`  
`[inline], [static]`

### Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

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

**26.2.6.19** `static uint32_t GPIO_PortGetInterruptFlags ( GPIO_Type * base )` `[inline],`  
`[static]`

### Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | GPIO base pointer. |
|-------------|--------------------|

### Return values

|                |                            |
|----------------|----------------------------|
| <i>current</i> | pin interrupt status flag. |
|----------------|----------------------------|

**26.2.6.20** `static uint32_t GPIO_GetPinsInterruptFlags ( GPIO_Type * base )` `[inline],`  
`[static]`

Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | GPIO base pointer. |
|-------------|--------------------|

Return values

|                |                            |
|----------------|----------------------------|
| <i>current</i> | pin interrupt status flag. |
|----------------|----------------------------|

**26.2.6.21 static void GPIO\_PortClearInterruptFlags ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**26.2.6.22 static void GPIO\_ClearPinsInterruptFlags ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |



## Chapter 27

# GPT: General Purpose Timer

### 27.1 Overview

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

### 27.2 Function groups

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

#### 27.2.1 Initialization and deinitialization

The function `GPT_Init()` initializes the gpt with specified configurations. The function `GPT_GetDefaultConfig()` gets the default configurations. The initialization function configures the restart/free-run mode and input selection when running.

The function `GPT_Deinit()` stops the timer and turns off the module clock.

### 27.3 Typical use case

#### 27.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/gpt`

### Data Structures

- struct `gpt_config_t`  
*Structure to configure the running mode. [More...](#)*

### Enumerations

- enum `gpt_clock_source_t` {  
    `kGPT_ClockSource_Off` = 0U,  
    `kGPT_ClockSource_Periph` = 1U,  
    `kGPT_ClockSource_HighFreq` = 2U,  
    `kGPT_ClockSource_Ext` = 3U,  
    `kGPT_ClockSource_LowFreq` = 4U,  
    `kGPT_ClockSource_Osc` = 5U }  
*List of clock sources.*

## Typical use case

- enum `gpt_input_capture_channel_t` {  
    `kGPT_InputCapture_Channel1` = 0U,  
    `kGPT_InputCapture_Channel2` = 1U }  
    *List of input capture channel number.*
- enum `gpt_input_operation_mode_t` {  
    `kGPT_InputOperation_Disabled` = 0U,  
    `kGPT_InputOperation_RiseEdge` = 1U,  
    `kGPT_InputOperation_FallEdge` = 2U,  
    `kGPT_InputOperation_BothEdge` = 3U }  
    *List of input capture operation mode.*
- enum `gpt_output_compare_channel_t` {  
    `kGPT_OutputCompare_Channel1` = 0U,  
    `kGPT_OutputCompare_Channel2` = 1U,  
    `kGPT_OutputCompare_Channel3` = 2U }  
    *List of output compare channel number.*
- enum `gpt_output_operation_mode_t` {  
    `kGPT_OutputOperation_Disconnected` = 0U,  
    `kGPT_OutputOperation_Toggle` = 1U,  
    `kGPT_OutputOperation_Clear` = 2U,  
    `kGPT_OutputOperation_Set` = 3U,  
    `kGPT_OutputOperation_Activelow` = 4U }  
    *List of output compare operation mode.*
- enum `gpt_interrupt_enable_t` {  
    `kGPT_OutputCompare1InterruptEnable` = GPT\_IR\_OF1IE\_MASK,  
    `kGPT_OutputCompare2InterruptEnable` = GPT\_IR\_OF2IE\_MASK,  
    `kGPT_OutputCompare3InterruptEnable` = GPT\_IR\_OF3IE\_MASK,  
    `kGPT_InputCapture1InterruptEnable` = GPT\_IR\_IF1IE\_MASK,  
    `kGPT_InputCapture2InterruptEnable` = GPT\_IR\_IF2IE\_MASK,  
    `kGPT_RollOverFlagInterruptEnable` = GPT\_IR\_ROVIE\_MASK }  
    *List of GPT interrupts.*
- enum `gpt_status_flag_t` {  
    `kGPT_OutputCompare1Flag` = GPT\_SR\_OF1\_MASK,  
    `kGPT_OutputCompare2Flag` = GPT\_SR\_OF2\_MASK,  
    `kGPT_OutputCompare3Flag` = GPT\_SR\_OF3\_MASK,  
    `kGPT_InputCapture1Flag` = GPT\_SR\_IF1\_MASK,  
    `kGPT_InputCapture2Flag` = GPT\_SR\_IF2\_MASK,  
    `kGPT_RollOverFlag` = GPT\_SR\_ROV\_MASK }  
    *Status flag.*

## Driver version

- #define `FSL_GPT_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
    *Version 2.0.0.*

## Initialization and deinitialization

- void `GPT_Init` (GPT\_Type \*base, const `gpt_config_t` \*initConfig)

- *Initialize GPT to reset state and initialize running mode.*
- void [GPT\\_Deinit](#) (GPT\_Type \*base)  
*Disables the module and gates the GPT clock.*
- void [GPT\\_GetDefaultConfig](#) (gpt\_config\_t \*config)  
*Fills in the GPT configuration structure with default settings.*

## Software Reset

- static void [GPT\\_SoftwareReset](#) (GPT\_Type \*base)  
*Software reset of GPT module.*

## Clock source and frequency control

- static void [GPT\\_SetClockSource](#) (GPT\_Type \*base, gpt\_clock\_source\_t source)  
*Set clock source of GPT.*
- static gpt\_clock\_source\_t [GPT\\_GetClockSource](#) (GPT\_Type \*base)  
*Get clock source of GPT.*
- static void [GPT\\_SetClockDivider](#) (GPT\_Type \*base, uint32\_t divider)  
*Set pre scaler of GPT.*
- static uint32\_t [GPT\\_GetClockDivider](#) (GPT\_Type \*base)  
*Get clock divider in GPT module.*
- static void [GPT\\_SetOscClockDivider](#) (GPT\_Type \*base, uint32\_t divider)  
*OSC 24M pre-scaler before selected by clock source.*
- static uint32\_t [GPT\\_GetOscClockDivider](#) (GPT\_Type \*base)  
*Get OSC 24M clock divider in GPT module.*

## Timer Start and Stop

- static void [GPT\\_StartTimer](#) (GPT\_Type \*base)  
*Start GPT timer.*
- static void [GPT\\_StopTimer](#) (GPT\_Type \*base)  
*Stop GPT timer.*

## Read the timer period

- static uint32\_t [GPT\\_GetCurrentTimerCount](#) (GPT\_Type \*base)  
*Reads the current GPT counting value.*

## GPT Input/Output Signal Control

- static void [GPT\\_SetInputOperationMode](#) (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel, gpt\_input\_operation\_mode\_t mode)  
*Set GPT operation mode of input capture channel.*
- static gpt\_input\_operation\_mode\_t [GPT\\_GetInputOperationMode](#) (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel)  
*Get GPT operation mode of input capture channel.*
- static uint32\_t [GPT\\_GetInputCaptureValue](#) (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel)  
*Get GPT input capture value of certain channel.*

## Data Structure Documentation

- static void [GPT\\_SetOutputOperationMode](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel, [gpt\\_output\\_operation\\_mode\\_t](#) mode)  
*Set GPT operation mode of output compare channel.*
- static [gpt\\_output\\_operation\\_mode\\_t](#) [GPT\\_GetOutputOperationMode](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)  
*Get GPT operation mode of output compare channel.*
- static void [GPT\\_SetOutputCompareValue](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel, uint32\_t value)  
*Set GPT output compare value of output compare channel.*
- static uint32\_t [GPT\\_GetOutputCompareValue](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)  
*Get GPT output compare value of output compare channel.*
- static void [GPT\\_ForceOutput](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)  
*Force GPT output action on output compare channel, ignoring comparator.*

## GPT Interrupt and Status Interface

- static void [GPT\\_EnableInterrupts](#) (GPT\_Type \*base, uint32\_t mask)  
*Enables the selected GPT interrupts.*
- static void [GPT\\_DisableInterrupts](#) (GPT\_Type \*base, uint32\_t mask)  
*Disables the selected GPT interrupts.*
- static uint32\_t [GPT\\_GetEnabledInterrupts](#) (GPT\_Type \*base)  
*Gets the enabled GPT interrupts.*

## Status Interface

- static uint32\_t [GPT\\_GetStatusFlags](#) (GPT\_Type \*base, [gpt\\_status\\_flag\\_t](#) flags)  
*Get GPT status flags.*
- static void [GPT\\_ClearStatusFlags](#) (GPT\_Type \*base, [gpt\\_status\\_flag\\_t](#) flags)  
*Clears the GPT status flags.*

## 27.4 Data Structure Documentation

### 27.4.1 struct [gpt\\_config\\_t](#)

#### Data Fields

- [gpt\\_clock\\_source\\_t](#) clockSource  
*clock source for GPT module.*
- uint32\_t divider  
*clock divider (prescaler+1) from clock source to counter.*
- bool [enableFreeRun](#)  
*true: FreeRun mode, false: Restart mode.*
- bool [enableRunInWait](#)  
*GPT enabled in wait mode.*
- bool [enableRunInStop](#)  
*GPT enabled in stop mode.*
- bool [enableRunInDoze](#)  
*GPT enabled in doze mode.*



- bool `enableRunInDbg`  
*GPT enabled in debug mode.*
- bool `enableMode`  
`true:` counter reset to 0 when enabled;  
`false:` counter retain its value when enabled.

### 27.4.1.0.0.7 Field Documentation

27.4.1.0.0.7.1 `gpt_clock_source_t gpt_config_t::clockSource`

27.4.1.0.0.7.2 `uint32_t gpt_config_t::divider`

27.4.1.0.0.7.3 `bool gpt_config_t::enableFreeRun`

27.4.1.0.0.7.4 `bool gpt_config_t::enableRunInWait`

27.4.1.0.0.7.5 `bool gpt_config_t::enableRunInStop`

27.4.1.0.0.7.6 `bool gpt_config_t::enableRunInDoze`

27.4.1.0.0.7.7 `bool gpt_config_t::enableRunInDbg`

27.4.1.0.0.7.8 `bool gpt_config_t::enableMode`

## 27.5 Enumeration Type Documentation

### 27.5.1 `enum gpt_clock_source_t`

Note

Actual number of clock sources is SoC dependent

Enumerator

*kGPT\_ClockSource\_Off* GPT Clock Source Off.

*kGPT\_ClockSource\_Periph* GPT Clock Source from Peripheral Clock.

*kGPT\_ClockSource\_HighFreq* GPT Clock Source from High Frequency Reference Clock.

*kGPT\_ClockSource\_Ext* GPT Clock Source from external pin.

*kGPT\_ClockSource\_LowFreq* GPT Clock Source from Low Frequency Reference Clock.

*kGPT\_ClockSource\_Osc* GPT Clock Source from Crystal oscillator.

### 27.5.2 `enum gpt_input_capture_channel_t`

Enumerator

*kGPT\_InputCapture\_Channel1* GPT Input Capture Channel1.

*kGPT\_InputCapture\_Channel2* GPT Input Capture Channel2.

## Enumeration Type Documentation

### 27.5.3 enum gpt\_input\_operation\_mode\_t

Enumerator

*kGPT\_InputOperation\_Disabled* Don't capture.  
*kGPT\_InputOperation\_RiseEdge* Capture on rising edge of input pin.  
*kGPT\_InputOperation\_FallEdge* Capture on falling edge of input pin.  
*kGPT\_InputOperation\_BothEdge* Capture on both edges of input pin.

### 27.5.4 enum gpt\_output\_compare\_channel\_t

Enumerator

*kGPT\_OutputCompare\_Channel1* Output Compare Channel1.  
*kGPT\_OutputCompare\_Channel2* Output Compare Channel2.  
*kGPT\_OutputCompare\_Channel3* Output Compare Channel3.

### 27.5.5 enum gpt\_output\_operation\_mode\_t

Enumerator

*kGPT\_OutputOperation\_Disconnected* Don't change output pin.  
*kGPT\_OutputOperation\_Toggle* Toggle output pin.  
*kGPT\_OutputOperation\_Clear* Set output pin low.  
*kGPT\_OutputOperation\_Set* Set output pin high.  
*kGPT\_OutputOperation\_Activelow* Generate a active low pulse on output pin.

### 27.5.6 enum gpt\_interrupt\_enable\_t

Enumerator

*kGPT\_OutputCompare1InterruptEnable* Output Compare Channel1 interrupt enable.  
*kGPT\_OutputCompare2InterruptEnable* Output Compare Channel2 interrupt enable.  
*kGPT\_OutputCompare3InterruptEnable* Output Compare Channel3 interrupt enable.  
*kGPT\_InputCapture1InterruptEnable* Input Capture Channel1 interrupt enable.  
*kGPT\_InputCapture2InterruptEnable* Input Capture Channel1 interrupt enable.  
*kGPT\_RollOverFlagInterruptEnable* Counter rolled over interrupt enable.

## 27.5.7 enum gpt\_status\_flag\_t

Enumerator

***kGPT\_OutputCompare1Flag*** Output compare channel 1 event.  
***kGPT\_OutputCompare2Flag*** Output compare channel 2 event.  
***kGPT\_OutputCompare3Flag*** Output compare channel 3 event.  
***kGPT\_InputCapture1Flag*** Input Capture channel 1 event.  
***kGPT\_InputCapture2Flag*** Input Capture channel 2 event.  
***kGPT\_RollOverFlag*** Counter reaches maximum value and rolled over to 0 event.

## 27.6 Function Documentation

### 27.6.1 void GPT\_Init ( GPT\_Type \* *base*, const gpt\_config\_t \* *initConfig* )

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | GPT peripheral base address.    |
| <i>initConfig</i> | GPT mode setting configuration. |

### 27.6.2 void GPT\_Deinit ( GPT\_Type \* *base* )

Parameters

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

### 27.6.3 void GPT\_GetDefaultConfig ( gpt\_config\_t \* *config* )

The default values are:

```
* config->clockSource = kGPT_ClockSource_Periph;
* config->divider = 1U;
* config->enableRunInStop = true;
* config->enableRunInWait = true;
* config->enableRunInDoze = false;
* config->enableRunInDbg = false;
* config->enableFreeRun = true;
* config->enableMode = true;
*
```

## Function Documentation

Parameters

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

**27.6.4 static void GPT\_SoftwareReset ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

**27.6.5 static void GPT\_SetClockSource ( GPT\_Type \* *base*, gpt\_clock\_source\_t *source* ) [inline], [static]**

Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | GPT peripheral base address.                                               |
| <i>source</i> | Clock source (see <a href="#">gpt_clock_source_t</a> typedef enumeration). |

**27.6.6 static gpt\_clock\_source\_t GPT\_GetClockSource ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

clock source (see [gpt\\_clock\\_source\\_t](#) typedef enumeration).

**27.6.7 static void GPT\_SetClockDivider ( GPT\_Type \* *base*, uint32\_t *divider* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | Divider of GPT (1-4096).     |

**27.6.8** `static uint32_t GPT_GetClockDivider ( GPT_Type * base ) [inline], [static]`

Parameters

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

Returns

clock divider in GPT module (1-4096).

**27.6.9** `static void GPT_SetOscClockDivider ( GPT_Type * base, uint32_t divider ) [inline], [static]`

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | OSC Divider(1-16).           |

**27.6.10** `static uint32_t GPT_GetOscClockDivider ( GPT_Type * base ) [inline], [static]`

Parameters

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

Returns

OSC clock divider in GPT module (1-16).

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

## Function Documentation

Parameters

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

**27.6.12 static void GPT\_StopTimer ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

**27.6.13 static uint32\_t GPT\_GetCurrentTimerCount ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

Current GPT counter value.

**27.6.14 static void GPT\_SetInputOperationMode ( GPT\_Type \* *base*, gpt\_input\_capture\_channel\_t *channel*, gpt\_input\_operation\_mode\_t *mode* ) [inline], [static]**

Parameters

|                |                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                           |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration).             |
| <i>mode</i>    | GPT input capture operation mode (see <a href="#">gpt_input_operation_mode_t</a> typedef enumeration). |

**27.6.15 static gpt\_input\_operation\_mode\_t GPT\_GetInputOperationMode ( GPT\_Type \* *base*, gpt\_input\_capture\_channel\_t *channel* ) [inline], [static]**

## Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                               |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration). |

## Returns

GPT input capture operation mode (see [gpt\\_input\\_operation\\_mode\\_t](#) typedef enumeration).

**27.6.16** `static uint32_t GPT_GetInputCaptureValue ( GPT_Type * base,  
gpt_input_capture_channel_t channel ) [inline], [static]`

## Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                               |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration). |

## Returns

GPT input capture value.

**27.6.17** `static void GPT_SetOutputOperationMode ( GPT_Type * base,  
gpt_output_compare_channel_t channel, gpt_output_operation_mode_t  
mode ) [inline], [static]`

## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |
| <i>mode</i>    | GPT output operation mode (see <a href="#">gpt_output_operation_mode_t</a> typedef enumeration).   |

**27.6.18** `static gpt_output_operation_mode_t GPT_GetOutputOperationMode (  
GPT_Type * base, gpt_output_compare_channel_t channel ) [inline],  
[static]`

## Function Documentation

### Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

### Returns

GPT output operation mode (see [gpt\\_output\\_operation\\_mode\\_t](#) typedef enumeration).

**27.6.19** `static void GPT_SetOutputCompareValue ( GPT_Type * base,  
gpt_output_compare_channel_t channel, uint32_t value ) [inline],  
[static]`

### Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |
| <i>value</i>   | GPT output compare value.                                                                          |

**27.6.20** `static uint32_t GPT_GetOutputCompareValue ( GPT_Type * base,  
gpt_output_compare_channel_t channel ) [inline], [static]`

### Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

### Returns

GPT output compare value.

**27.6.21** `static void GPT_ForceOutput ( GPT_Type * base, gpt_output_compare_ -  
channel_t channel ) [inline], [static]`



## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

### 27.6.22 static void GPT\_EnableInterrupts ( GPT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | GPT peripheral base address.                                                                                        |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">gpt_interrupt_enable_t</a> |

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

## Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | GPT peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">gpt_interrupt_enable_t</a> |

### 27.6.24 static uint32\_t GPT\_GetEnabledInterrupts ( GPT\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

The enabled interrupts. This is the logical OR of members of the enumeration [gpt\\_interrupt\\_enable\\_t](#)

## Function Documentation

**27.6.25** `static uint32_t GPT_GetStatusFlags ( GPT_Type * base, gpt_status_flag_t flags ) [inline], [static]`

## Parameters

|              |                                                                                  |
|--------------|----------------------------------------------------------------------------------|
| <i>base</i>  | GPT peripheral base address.                                                     |
| <i>flags</i> | GPT status flag mask (see <a href="#">gpt_status_flag_t</a> for bit definition). |

## Returns

GPT status, each bit represents one status flag.

**27.6.26** `static void GPT_ClearStatusFlags ( GPT_Type * base, gpt_status_flag_t flags ) [inline], [static]`

## Parameters

|              |                                                                                  |
|--------------|----------------------------------------------------------------------------------|
| <i>base</i>  | GPT peripheral base address.                                                     |
| <i>flags</i> | GPT status flag mask (see <a href="#">gpt_status_flag_t</a> for bit definition). |



## Chapter 28

# KPP: KeyPad Port Driver

### 28.1 Overview

The MCUXpresso SDK provides a peripheral driver for the KeyPad Port block of MCUXpresso SDK devices.

The KPP Initialize is to initialize for common configure: gate the KPP clock, configure columns, and rows features. The KPP Deinitialize is to ungate the clock.

The KPP provide the function to enable/disable interrupts. The KPP provide key press scanning function `KPP_keyPressScanning`. This API should be called by the Interrupt handler in application. KPP still provides functions to get and clear status flags.

### 28.2 Typical use case

#### Data Structures

- struct `kpp_config_t`  
*Lists of KPP status. [More...](#)*

#### Enumerations

- enum `kpp_interrupt_enable_t` {  
`kKPP_keyDepressInterrupt` = `KPP_KPSR_KDIE_MASK`,  
`kKPP_keyReleaseInterrupt` = `KPP_KPSR_KRIE_MASK` }  
*List of interrupts supported by the peripheral.*
- enum `kpp_sync_operation_t` {  
`kKPP_ClearKeyDepressSyncChain` = `KPP_KPSR_KDSC_MASK`,  
`kKPP_SetKeyReleasesSyncChain` = `KPP_KPSR_KRSS_MASK` }  
*Lists of KPP synchronize chain operation.*

#### Driver version

- #define `FSL_KPP_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
*KPP driver version 2.0.0.*

#### Initialization and De-initialization

- void `KPP_Init` (`KPP_Type *base`, `kpp_config_t *configure`)  
*KPP initialize.*
- void `KPP_Deinit` (`KPP_Type *base`)  
*Deinitializes the KPP module and gates the clock.*

## Enumeration Type Documentation

### KPP Basic Operation

- static void [KPP\\_EnableInterrupts](#) (KPP\_Type \*base, uint16\_t mask)  
*Enable the interrupt.*
- static void [KPP\\_DisableInterrupts](#) (KPP\_Type \*base, uint16\_t mask)  
*Disable the interrupt.*
- static uint16\_t [KPP\\_GetStatusFlag](#) (KPP\_Type \*base)  
*Gets the KPP interrupt event status.*
- static void [KPP\\_ClearStatusFlag](#) (KPP\_Type \*base, uint16\_t mask)  
*Clears KPP status flag.*
- static void [KPP\\_SetSynchronizeChain](#) (KPP\_Type \*base, uint16\_t mask)  
*Set KPP synchronization chain.*
- void [KPP\\_keyPressScanning](#) (KPP\_Type \*base, uint8\_t \*data, uint32\_t clockSrc\_Hz)  
*Keypad press scanning.*

## 28.3 Data Structure Documentation

### 28.3.1 struct kpp\_config\_t

#### Data Fields

- uint8\_t [activeRow](#)  
*The row number: bit 7 ~ 0 represents the row 7 ~ 0.*
- uint8\_t [activeColumn](#)  
*The column number: bit 7 ~ 0 represents the column 7 ~ 0.*
- uint16\_t [interrupt](#)  
*KPP interrupt source.*

#### 28.3.1.0.0.8 Field Documentation

##### 28.3.1.0.0.8.1 uint8\_t kpp\_config\_t::activeRow

##### 28.3.1.0.0.8.2 uint8\_t kpp\_config\_t::activeColumn

##### 28.3.1.0.0.8.3 uint16\_t kpp\_config\_t::interrupt

A logical OR of "kpp\_interrupt\_enable\_t".

## 28.4 Macro Definition Documentation

### 28.4.1 #define FSL\_KPP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 28.5 Enumeration Type Documentation

### 28.5.1 enum kpp\_interrupt\_enable\_t

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

Enumerator

- kKPP\_keyDepressInterrupt* Keypad depress interrupt source.
- kKPP\_keyReleaseInterrupt* Keypad release interrupt source.

## 28.5.2 enum kpp\_sync\_operation\_t

Enumerator

- kKPP\_ClearKeyDepressSyncChain* Keypad depress interrupt status.
- kKPP\_SetKeyReleasesSyncChain* Keypad release interrupt status.

## 28.6 Function Documentation

### 28.6.1 void KPP\_Init ( KPP\_Type \* *base*, kpp\_config\_t \* *configure* )

This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | KPP peripheral base address.             |
| <i>configure</i> | The KPP configuration structure pointer. |

### 28.6.2 void KPP\_Deinit ( KPP\_Type \* *base* )

This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

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

### 28.6.3 static void KPP\_EnableInterrupts ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

Parameters

---

## Function Documentation

|             |                                                                                              |
|-------------|----------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                                 |
| <i>mask</i> | KPP interrupts to enable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |

### 28.6.4 static void KPP\_DisableInterrupts ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                                  |
| <i>mask</i> | KPP interrupts to disable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |

### 28.6.5 static uint16\_t KPP\_GetStatusFlag ( KPP\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The status of the KPP. Application can use the enum type in the "kpp\_interrupt\_enable\_t" to get the right status of the related event.

### 28.6.6 static void KPP\_ClearStatusFlag ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                               |
| <i>mask</i> | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |

### 28.6.7 static void KPP\_SetSynchronizeChain ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]



## Parameters

|             |                                                                                          |
|-------------|------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                             |
| <i>mask</i> | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_sync_operation_t. |

### 28.6.8 void KPP\_keyPressScanning ( KPP\_Type \* *base*, uint8\_t \* *data*, uint32\_t *clockSrc\_Hz* )

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

## Parameters

|             |                                                                                                                                                                                                                                                                                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                                                                                                                                                                                                                                                                |
| <i>data</i> | KPP key press scanning data. The data buffer should be prepared with length at least equal to KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX. the data pointer is recommended to be a array like uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]. for example the data[2] = 4, that means in column 1 row 2 has a key press event. |



## Chapter 29

# LPI2C: Low Power I2C Driver

### 29.1 Overview

#### Modules

- [LPI2C FreeRTOS Driver](#)
- [LPI2C Master DMA Driver](#)
- [LPI2C Master Driver](#)
- [LPI2C Slave Driver](#)

#### Macros

- `#define LPI2C_WAIT_TIMEOUT 0U` /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Timeout times for waiting flag.*

#### Enumerations

- `enum _lpi2c_status` {  
    `kStatus_LPI2C_Busy` = MAKE\_STATUS(kStatusGroup\_LPI2C, 0),  
    `kStatus_LPI2C_Idle` = MAKE\_STATUS(kStatusGroup\_LPI2C, 1),  
    `kStatus_LPI2C_Nak` = MAKE\_STATUS(kStatusGroup\_LPI2C, 2),  
    `kStatus_LPI2C_FifoError` = MAKE\_STATUS(kStatusGroup\_LPI2C, 3),  
    `kStatus_LPI2C_BitError` = MAKE\_STATUS(kStatusGroup\_LPI2C, 4),  
    `kStatus_LPI2C_ArbitrationLost` = MAKE\_STATUS(kStatusGroup\_LPI2C, 5),  
    `kStatus_LPI2C_PinLowTimeout`,  
    `kStatus_LPI2C_NoTransferInProgress`,  
    `kStatus_LPI2C_DmaRequestFail` = MAKE\_STATUS(kStatusGroup\_LPI2C, 8),  
    `kStatus_LPI2C_Timeout` = MAKE\_STATUS(kStatusGroup\_LPI2C, 9) }  
*LPI2C status return codes.*

#### Driver version

- `#define FSL_LPI2C_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 5))  
*LPI2C driver version 2.1.5.*

### 29.2 Macro Definition Documentation

#### 29.2.1 `#define FSL_LPI2C_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 5))

#### 29.2.2 `#define LPI2C_WAIT_TIMEOUT 0U` /\* Define to zero means keep waiting until the flag is assert/deassert. \*/

## Enumeration Type Documentation

### 29.3 Enumeration Type Documentation

#### 29.3.1 enum `_lpi2c_status`

Enumerator

*kStatus\_LPI2C\_Busy* The master is already performing a transfer.

*kStatus\_LPI2C\_Idle* The slave driver is idle.

*kStatus\_LPI2C\_Nak* The slave device sent a NAK in response to a byte.

*kStatus\_LPI2C\_FifoError* FIFO under run or overrun.

*kStatus\_LPI2C\_BitError* Transferred bit was not seen on the bus.

*kStatus\_LPI2C\_ArbitrationLost* Arbitration lost error.

*kStatus\_LPI2C\_PinLowTimeout* SCL or SDA were held low longer than the timeout.

*kStatus\_LPI2C\_NoTransferInProgress* Attempt to abort a transfer when one is not in progress.

*kStatus\_LPI2C\_DmaRequestFail* DMA request failed.

*kStatus\_LPI2C\_Timeout* Timeout polling status flags.

## 29.4 LPI2C Master Driver

### 29.4.1 Overview

#### Data Structures

- struct `lpi2c_master_config_t`  
*Structure with settings to initialize the LPI2C master module. [More...](#)*
- struct `lpi2c_data_match_config_t`  
*LPI2C master data match configuration structure. [More...](#)*
- struct `lpi2c_master_transfer_t`  
*Non-blocking transfer descriptor structure. [More...](#)*
- struct `lpi2c_master_handle_t`  
*Driver handle for master non-blocking APIs. [More...](#)*

#### Typedefs

- typedef `void(* lpi2c_master_transfer_callback_t )(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)`  
*Master completion callback function pointer type.*

#### Enumerations

- enum `_lpi2c_master_flags` {  
`kLPI2C_MasterTxReadyFlag = LPI2C_MSR_TDF_MASK,`  
`kLPI2C_MasterRxReadyFlag = LPI2C_MSR_RDF_MASK,`  
`kLPI2C_MasterEndOfPacketFlag = LPI2C_MSR_EPF_MASK,`  
`kLPI2C_MasterStopDetectFlag = LPI2C_MSR_SDF_MASK,`  
`kLPI2C_MasterNackDetectFlag = LPI2C_MSR_NDF_MASK,`  
`kLPI2C_MasterArbitrationLostFlag = LPI2C_MSR_ALF_MASK,`  
`kLPI2C_MasterFifoErrFlag = LPI2C_MSR_FEF_MASK,`  
`kLPI2C_MasterPinLowTimeoutFlag = LPI2C_MSR_PLTF_MASK,`  
`kLPI2C_MasterDataMatchFlag = LPI2C_MSR_DMF_MASK,`  
`kLPI2C_MasterBusyFlag = LPI2C_MSR_MBF_MASK,`  
`kLPI2C_MasterBusBusyFlag = LPI2C_MSR_BBF_MASK }`  
*LPI2C master peripheral flags.*
- enum `lpi2c_direction_t` {  
`kLPI2C_Write = 0U,`  
`kLPI2C_Read = 1U }`  
*Direction of master and slave transfers.*
- enum `lpi2c_master_pin_config_t` {

## LPI2C Master Driver

```
kLPI2C_2PinOpenDrain = 0x0U,  
kLPI2C_2PinOutputOnly = 0x1U,  
kLPI2C_2PinPushPull = 0x2U,  
kLPI2C_4PinPushPull = 0x3U,  
kLPI2C_2PinOpenDrainWithSeparateSlave,  
kLPI2C_2PinOutputOnlyWithSeparateSlave,  
kLPI2C_2PinPushPullWithSeparateSlave,  
kLPI2C_4PinPushPullWithInvertedOutput = 0x7U }
```

*LPI2C pin configuration.*

- enum `lpi2c_host_request_source_t` {  
    `kLPI2C_HostRequestExternalPin` = 0x0U,  
    `kLPI2C_HostRequestInputTrigger` = 0x1U }

*LPI2C master host request selection.*

- enum `lpi2c_host_request_polarity_t` {  
    `kLPI2C_HostRequestPinActiveLow` = 0x0U,  
    `kLPI2C_HostRequestPinActiveHigh` = 0x1U }

*LPI2C master host request pin polarity configuration.*

- enum `lpi2c_data_match_config_mode_t` {  
    `kLPI2C_MatchDisabled` = 0x0U,  
    `kLPI2C_1stWordEqualsM0OrM1` = 0x2U,  
    `kLPI2C_AnyWordEqualsM0OrM1` = 0x3U,  
    `kLPI2C_1stWordEqualsM0And2ndWordEqualsM1`,  
    `kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1`,  
    `kLPI2C_1stWordAndM1EqualsM0AndM1`,  
    `kLPI2C_AnyWordAndM1EqualsM0AndM1` }

*LPI2C master data match configuration modes.*

- enum `_lpi2c_master_transfer_flags` {  
    `kLPI2C_TransferDefaultFlag` = 0x00U,  
    `kLPI2C_TransferNoStartFlag` = 0x01U,  
    `kLPI2C_TransferRepeatedStartFlag` = 0x02U,  
    `kLPI2C_TransferNoStopFlag` = 0x04U }

*Transfer option flags.*

## Initialization and deinitialization

- void `LPI2C_MasterGetDefaultConfig` (`lpi2c_master_config_t` \*masterConfig)  
*Provides a default configuration for the LPI2C master peripheral.*
- void `LPI2C_MasterInit` (`LPI2C_Type` \*base, const `lpi2c_master_config_t` \*masterConfig, `uint32_t` sourceClock\_Hz)  
*Initializes the LPI2C master peripheral.*
- void `LPI2C_MasterDeinit` (`LPI2C_Type` \*base)  
*Deinitializes the LPI2C master peripheral.*
- void `LPI2C_MasterConfigureDataMatch` (`LPI2C_Type` \*base, const `lpi2c_data_match_config_t` \*config)  
*Configures LPI2C master data match feature.*
- status\_t `LPI2C_MasterCheckAndClearError` (`LPI2C_Type` \*base, `uint32_t` status)

- status\_t **LPI2C\_CheckForBusyBus** (LPI2C\_Type \*base)
- static void **LPI2C\_MasterReset** (LPI2C\_Type \*base)  
*Performs a software reset.*
- static void **LPI2C\_MasterEnable** (LPI2C\_Type \*base, bool enable)  
*Enables or disables the LPI2C module as master.*

## Status

- static uint32\_t **LPI2C\_MasterGetStatusFlags** (LPI2C\_Type \*base)  
*Gets the LPI2C master status flags.*
- static void **LPI2C\_MasterClearStatusFlags** (LPI2C\_Type \*base, uint32\_t statusMask)  
*Clears the LPI2C master status flag state.*

## Interrupts

- static void **LPI2C\_MasterEnableInterrupts** (LPI2C\_Type \*base, uint32\_t interruptMask)  
*Enables the LPI2C master interrupt requests.*
- static void **LPI2C\_MasterDisableInterrupts** (LPI2C\_Type \*base, uint32\_t interruptMask)  
*Disables the LPI2C master interrupt requests.*
- static uint32\_t **LPI2C\_MasterGetEnabledInterrupts** (LPI2C\_Type \*base)  
*Returns the set of currently enabled LPI2C master interrupt requests.*

## DMA control

- static void **LPI2C\_MasterEnableDMA** (LPI2C\_Type \*base, bool enableTx, bool enableRx)  
*Enables or disables LPI2C master DMA requests.*
- static uint32\_t **LPI2C\_MasterGetTxFifoAddress** (LPI2C\_Type \*base)  
*Gets LPI2C master transmit data register address for DMA transfer.*
- static uint32\_t **LPI2C\_MasterGetRxFifoAddress** (LPI2C\_Type \*base)  
*Gets LPI2C master receive data register address for DMA transfer.*

## FIFO control

- static void **LPI2C\_MasterSetWatermarks** (LPI2C\_Type \*base, size\_t txWords, size\_t rxWords)  
*Sets the watermarks for LPI2C master FIFOs.*
- static void **LPI2C\_MasterGetFifoCounts** (LPI2C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)  
*Gets the current number of words in the LPI2C master FIFOs.*

## Bus operations

- void **LPI2C\_MasterSetBaudRate** (LPI2C\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t baudRate\_Hz)  
*Sets the I2C bus frequency for master transactions.*
- static bool **LPI2C\_MasterGetBusIdleState** (LPI2C\_Type \*base)

## LPI2C Master Driver

- Returns whether the bus is idle.*
- status\_t [LPI2C\\_MasterStart](#) (LPI2C\_Type \*base, uint8\_t address, [lpi2c\\_direction\\_t](#) dir)  
*Sends a START signal and slave address on the I2C bus.*
- static status\_t [LPI2C\\_MasterRepeatedStart](#) (LPI2C\_Type \*base, uint8\_t address, [lpi2c\\_direction\\_t](#) dir)  
*Sends a repeated START signal and slave address on the I2C bus.*
- status\_t [LPI2C\\_MasterSend](#) (LPI2C\_Type \*base, const void \*txBuff, size\_t txSize)  
*Performs a polling send transfer on the I2C bus.*
- status\_t [LPI2C\\_MasterReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize)  
*Performs a polling receive transfer on the I2C bus.*
- status\_t [LPI2C\\_MasterStop](#) (LPI2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- status\_t [LPI2C\\_MasterTransferBlocking](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_transfer\\_t](#) \*transfer)  
*Performs a master polling transfer on the I2C bus.*

## Non-blocking

- void [LPI2C\\_MasterTransferCreateHandle](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle, [lpi2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the LPI2C master non-blocking APIs.*
- status\_t [LPI2C\\_MasterTransferNonBlocking](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle, [lpi2c\\_master\\_transfer\\_t](#) \*transfer)  
*Performs a non-blocking transaction on the I2C bus.*
- status\_t [LPI2C\\_MasterTransferGetCount](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- void [LPI2C\\_MasterTransferAbort](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle)  
*Terminates a non-blocking LPI2C master transmission early.*

## IRQ handler

- void [LPI2C\\_MasterTransferHandleIRQ](#) (LPI2C\_Type \*base, [lpi2c\\_master\\_handle\\_t](#) \*handle)  
*Reusable routine to handle master interrupts.*

## 29.4.2 Data Structure Documentation

### 29.4.2.1 struct [lpi2c\\_master\\_config\\_t](#)

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_MasterGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.



## Data Fields

- bool `enableMaster`  
*Whether to enable master mode.*
- bool `enableDoze`  
*Whether master is enabled in doze mode.*
- bool `debugEnable`  
*Enable transfers to continue when halted in debug mode.*
- bool `ignoreAck`  
*Whether to ignore ACK/NACK.*
- `lpi2c_master_pin_config_t` `pinConfig`  
*The pin configuration option.*
- `uint32_t` `baudRate_Hz`  
*Desired baud rate in Hertz.*
- `uint32_t` `busIdleTimeout_ns`  
*Bus idle timeout in nanoseconds.*
- `uint32_t` `pinLowTimeout_ns`  
*Pin low timeout in nanoseconds.*
- `uint8_t` `sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SDA pin.*
- `uint8_t` `sclGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SCL pin.*
- struct {
  - bool `enable`  
*Enable host request.*
  - `lpi2c_host_request_source_t` `source`  
*Host request source.*
  - `lpi2c_host_request_polarity_t` `polarity`  
*Host request pin polarity.*
- } `hostRequest`  
  
*Host request options.*

### 29.4.2.1.0.9 Field Documentation

29.4.2.1.0.9.1 `bool lpi2c_master_config_t::enableMaster`

29.4.2.1.0.9.2 `bool lpi2c_master_config_t::enableDoze`

29.4.2.1.0.9.3 `bool lpi2c_master_config_t::debugEnable`

29.4.2.1.0.9.4 `bool lpi2c_master_config_t::ignoreAck`

29.4.2.1.0.9.5 `lpi2c_master_pin_config_t lpi2c_master_config_t::pinConfig`

29.4.2.1.0.9.6 `uint32_t lpi2c_master_config_t::baudRate_Hz`

29.4.2.1.0.9.7 `uint32_t lpi2c_master_config_t::busIdleTimeout_ns`

Set to 0 to disable.

## LPI2C Master Driver

**29.4.2.1.0.9.8** `uint32_t lpi2c_master_config_t::pinLowTimeout_ns`

Set to 0 to disable.

**29.4.2.1.0.9.9** `uint8_t lpi2c_master_config_t::sdaGlitchFilterWidth_ns`

Set to 0 to disable.

**29.4.2.1.0.9.10** `uint8_t lpi2c_master_config_t::sclGlitchFilterWidth_ns`

Set to 0 to disable.

**29.4.2.1.0.9.11** `bool lpi2c_master_config_t::enable`

**29.4.2.1.0.9.12** `lpi2c_host_request_source_t lpi2c_master_config_t::source`

**29.4.2.1.0.9.13** `lpi2c_host_request_polarity_t lpi2c_master_config_t::polarity`

**29.4.2.1.0.9.14** `struct { ... } lpi2c_master_config_t::hostRequest`

**29.4.2.2** `struct lpi2c_data_match_config_t`

### Data Fields

- `lpi2c_data_match_config_mode_t matchMode`  
*Data match configuration setting.*
- `bool rxDataMatchOnly`  
*When set to true, received data is ignored until a successful match.*
- `uint32_t match0`  
*Match value 0.*
- `uint32_t match1`  
*Match value 1.*

**29.4.2.2.0.10** Field Documentation

**29.4.2.2.0.10.1** `lpi2c_data_match_config_mode_t lpi2c_data_match_config_t::matchMode`

**29.4.2.2.0.10.2** `bool lpi2c_data_match_config_t::rxDataMatchOnly`

**29.4.2.2.0.10.3** `uint32_t lpi2c_data_match_config_t::match0`

**29.4.2.2.0.10.4** `uint32_t lpi2c_data_match_config_t::match1`

**29.4.2.3** `struct _lpi2c_master_transfer`

This structure is used to pass transaction parameters to the [LPI2C\\_MasterTransferNonBlocking\(\)](#) API.

### Data Fields

- `uint32_t flags`

- *Bit mask of options for the transfer.*  
uint16\_t [slaveAddress](#)  
*The 7-bit slave address.*
- [lpi2c\\_direction\\_t direction](#)  
*Either `kLPI2C_Read` or `kLPI2C_Write`.*
- uint32\_t [subaddress](#)  
*Sub address.*
- size\_t [subaddressSize](#)  
*Length of sub address to send in bytes.*
- void \* [data](#)  
*Pointer to data to transfer.*
- size\_t [dataSize](#)  
*Number of bytes to transfer.*

#### 29.4.2.3.0.11 Field Documentation

##### 29.4.2.3.0.11.1 uint32\_t lpi2c\_master\_transfer\_t::flags

See enumeration [\\_lpi2c\\_master\\_transfer\\_flags](#) for available options. Set to 0 or [kLPI2C\\_TransferDefaultFlag](#) for normal transfers.

##### 29.4.2.3.0.11.2 uint16\_t lpi2c\_master\_transfer\_t::slaveAddress

##### 29.4.2.3.0.11.3 lpi2c\_direction\_t lpi2c\_master\_transfer\_t::direction

##### 29.4.2.3.0.11.4 uint32\_t lpi2c\_master\_transfer\_t::subaddress

Transferred MSB first.

##### 29.4.2.3.0.11.5 size\_t lpi2c\_master\_transfer\_t::subaddressSize

Maximum size is 4 bytes.

##### 29.4.2.3.0.11.6 void\* lpi2c\_master\_transfer\_t::data

##### 29.4.2.3.0.11.7 size\_t lpi2c\_master\_transfer\_t::dataSize

#### 29.4.2.4 struct \_lpi2c\_master\_handle

Note

The contents of this structure are private and subject to change.

#### Data Fields

- uint8\_t [state](#)  
*Transfer state machine current state.*
- uint16\_t [remainingBytes](#)  
*Remaining byte count in current state.*
- uint8\_t \* [buf](#)

## LPI2C Master Driver

- *Buffer pointer for current state.*  
uint16\_t [commandBuffer](#) [7]
- *LPI2C command sequence.*  
lpi2c\_master\_transfer\_t [transfer](#)
- *Copy of the current transfer info.*  
lpi2c\_master\_transfer\_callback\_t [completionCallback](#)
- *Callback function pointer.*  
void \* [userData](#)
- *Application data passed to callback.*

### 29.4.2.4.0.12 Field Documentation

29.4.2.4.0.12.1 uint8\_t lpi2c\_master\_handle\_t::state

29.4.2.4.0.12.2 uint16\_t lpi2c\_master\_handle\_t::remainingBytes

29.4.2.4.0.12.3 uint8\_t\* lpi2c\_master\_handle\_t::buf

29.4.2.4.0.12.4 uint16\_t lpi2c\_master\_handle\_t::commandBuffer[7]

29.4.2.4.0.12.5 lpi2c\_master\_transfer\_t lpi2c\_master\_handle\_t::transfer

29.4.2.4.0.12.6 lpi2c\_master\_transfer\_callback\_t lpi2c\_master\_handle\_t::completionCallback

29.4.2.4.0.12.7 void\* lpi2c\_master\_handle\_t::userData

### 29.4.3 Typedef Documentation

29.4.3.1 typedef void(\* lpi2c\_master\_transfer\_callback\_t)(LPI2C\_Type \*base,  
lpi2c\_master\_handle\_t \*handle, status\_t completionStatus, void \*userData)

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [LPI2C\\_MasterTransferCreateHandle\(\)](#).

Parameters

|                          |                                                                                 |
|--------------------------|---------------------------------------------------------------------------------|
| <i>base</i>              | The LPI2C peripheral base address.                                              |
| <i>completion-Status</i> | Either #kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                      |

### 29.4.4 Enumeration Type Documentation

29.4.4.1 enum [lpi2c\\_master\\_flags](#)

The following status register flags can be cleared:

- `kLPI2C_MasterEndOfPacketFlag`
- `kLPI2C_MasterStopDetectFlag`
- `kLPI2C_MasterNackDetectFlag`
- `kLPI2C_MasterArbitrationLostFlag`
- `kLPI2C_MasterFifoErrFlag`
- `kLPI2C_MasterPinLowTimeoutFlag`
- `kLPI2C_MasterDataMatchFlag`

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

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

- `kLPI2C_MasterTxReadyFlag`*** Transmit data flag.
- `kLPI2C_MasterRxReadyFlag`*** Receive data flag.
- `kLPI2C_MasterEndOfPacketFlag`*** End Packet flag.
- `kLPI2C_MasterStopDetectFlag`*** Stop detect flag.
- `kLPI2C_MasterNackDetectFlag`*** NACK detect flag.
- `kLPI2C_MasterArbitrationLostFlag`*** Arbitration lost flag.
- `kLPI2C_MasterFifoErrFlag`*** FIFO error flag.
- `kLPI2C_MasterPinLowTimeoutFlag`*** Pin low timeout flag.
- `kLPI2C_MasterDataMatchFlag`*** Data match flag.
- `kLPI2C_MasterBusyFlag`*** Master busy flag.
- `kLPI2C_MasterBusBusyFlag`*** Bus busy flag.

#### 29.4.4.2 enum `lpi2c_direction_t`

Enumerator

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

#### 29.4.4.3 enum `lpi2c_master_pin_config_t`

Enumerator

- `kLPI2C_2PinOpenDrain`*** LPI2C Configured for 2-pin open drain mode.
- `kLPI2C_2PinOutputOnly`*** LPI2C Configured for 2-pin output only mode (ultra-fast mode)
- `kLPI2C_2PinPushPull`*** LPI2C Configured for 2-pin push-pull mode.
- `kLPI2C_4PinPushPull`*** LPI2C Configured for 4-pin push-pull mode.
- `kLPI2C_2PinOpenDrainWithSeparateSlave`*** LPI2C Configured for 2-pin open drain mode with separate LPI2C slave.

## LPI2C Master Driver

***kLPI2C\_2PinOutputOnlyWithSeparateSlave*** LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave.

***kLPI2C\_2PinPushPullWithSeparateSlave*** LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave.

***kLPI2C\_4PinPushPullWithInvertedOutput*** LPI2C Configured for 4-pin push-pull mode(inverted outputs)

### 29.4.4.4 enum lpi2c\_host\_request\_source\_t

Enumerator

***kLPI2C\_HostRequestExternalPin*** Select the LPI2C\_HREQ pin as the host request input.

***kLPI2C\_HostRequestInputTrigger*** Select the input trigger as the host request input.

### 29.4.4.5 enum lpi2c\_host\_request\_polarity\_t

Enumerator

***kLPI2C\_HostRequestPinActiveLow*** Configure the LPI2C\_HREQ pin active low.

***kLPI2C\_HostRequestPinActiveHigh*** Configure the LPI2C\_HREQ pin active high.

### 29.4.4.6 enum lpi2c\_data\_match\_config\_mode\_t

Enumerator

***kLPI2C\_MatchDisabled*** LPI2C Match Disabled.

***kLPI2C\_1stWordEqualsM0OrM1*** LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1.

***kLPI2C\_AnyWordEqualsM0OrM1*** LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1.

***kLPI2C\_1stWordEqualsM0And2ndWordEqualsM1*** LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1.

***kLPI2C\_AnyWordEqualsM0AndNextWordEqualsM1*** LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1.

***kLPI2C\_1stWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1.

***kLPI2C\_AnyWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1.

### 29.4.4.7 enum \_lpi2c\_master\_transfer\_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Enumerator

***kLPI2C\_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.

***kLPI2C\_TransferNoStartFlag*** Don't send a start condition, address, and sub address.

***kLPI2C\_TransferRepeatedStartFlag*** Send a repeated start condition.

***kLPI2C\_TransferNoStopFlag*** Don't send a stop condition.

## 29.4.5 Function Documentation

### 29.4.5.1 void LPI2C\_MasterGetDefaultConfig ( lpi2c\_master\_config\_t \* masterConfig )

This function provides the following default configuration for the LPI2C master peripheral:

```
* masterConfig->enableMaster           = true;
* masterConfig->debugEnable            = false;
* masterConfig->ignoreAck              = false;
* masterConfig->pinConfig              = kLPI2C_2PinOpenDrain;
* masterConfig->baudRate_Hz            = 100000U;
* masterConfig->busIdleTimeout_ns      = 0;
* masterConfig->pinLowTimeout_ns       = 0;
* masterConfig->sdaGlitchFilterWidth_ns = 0;
* masterConfig->sclGlitchFilterWidth_ns = 0;
* masterConfig->hostRequest.enable     = false;
* masterConfig->hostRequest.source     = kLPI2C_HostRequestExternalPin;
* masterConfig->hostRequest.polarity  = kLPI2C_HostRequestPinActiveHigh;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `LPI2C_MasterInit()`.

Parameters

|     |                     |                                                                                                         |
|-----|---------------------|---------------------------------------------------------------------------------------------------------|
| out | <i>masterConfig</i> | User provided configuration structure for default values. Refer to <code>lpi2c_master_config_t</code> . |
|-----|---------------------|---------------------------------------------------------------------------------------------------------|

### 29.4.5.2 void LPI2C\_MasterInit ( LPI2C\_Type \* base, const lpi2c\_master\_config\_t \* masterConfig, uint32\_t sourceClock\_Hz )

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

## LPI2C Master Driver

### Parameters

|                       |                                                                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.                                                                                                         |
| <i>masterConfig</i>   | User provided peripheral configuration. Use <a href="#">LPI2C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.            |

### 29.4.5.3 void LPI2C\_MasterDeinit ( LPI2C\_Type \* *base* )

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

### Parameters

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

### 29.4.5.4 void LPI2C\_MasterConfigureDataMatch ( LPI2C\_Type \* *base*, const *lpi2c\_data\_match\_config\_t* \* *config* )

### Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.   |
| <i>config</i> | Settings for the data match feature. |

### 29.4.5.5 static void LPI2C\_MasterReset ( LPI2C\_Type \* *base* ) [inline], [static]

Restores the LPI2C master peripheral to reset conditions.

### Parameters

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

### 29.4.5.6 static void LPI2C\_MasterEnable ( LPI2C\_Type \* *base*, bool *enable* ) [inline], [static]



## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                     |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as master. |

#### 29.4.5.7 `static uint32_t LPI2C_MasterGetStatusFlags ( LPI2C_Type * base ) [inline], [static]`

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

## Parameters

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

## Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

## See Also

[\\_lpi2c\\_master\\_flags](#)

#### 29.4.5.8 `static void LPI2C_MasterClearStatusFlags ( LPI2C_Type * base, uint32_t statusMask ) [inline], [static]`

The following status register flags can be cleared:

- [kLPI2C\\_MasterEndOfPacketFlag](#)
- [kLPI2C\\_MasterStopDetectFlag](#)
- [kLPI2C\\_MasterNackDetectFlag](#)
- [kLPI2C\\_MasterArbitrationLostFlag](#)
- [kLPI2C\\_MasterFifoErrFlag](#)
- [kLPI2C\\_MasterPinLowTimeoutFlag](#)
- [kLPI2C\\_MasterDataMatchFlag](#)

Attempts to clear other flags has no effect.

## LPI2C Master Driver

### Parameters

|                   |                                                                                                                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                                    |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_lpi2c_master_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_MasterGetStatusFlags()</a> . |

### See Also

[\\_lpi2c\\_master\\_flags](#).

#### 29.4.5.9 **static void LPI2C\_MasterEnableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be enabled as interrupts.

### Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 29.4.5.10 **static void LPI2C\_MasterDisableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be enabled as interrupts.

### Parameters

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

#### 29.4.5.11 **static uint32\_t LPI2C\_MasterGetEnabledInterrupts ( LPI2C\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

**29.4.5.12** `static void LPI2C_MasterEnableDMA ( LPI2C_Type * base, bool enableTx, bool enableRx ) [inline], [static]`

## Parameters

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.                                             |
| <i>enableTx</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |
| <i>enableRx</i> | Enable flag for receive DMA request. Pass true for enable, false for disable.  |

**29.4.5.13** `static uint32_t LPI2C_MasterGetTxFifoAddress ( LPI2C_Type * base ) [inline], [static]`

## Parameters

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

## Returns

The LPI2C Master Transmit Data Register address.

**29.4.5.14** `static uint32_t LPI2C_MasterGetRxFifoAddress ( LPI2C_Type * base ) [inline], [static]`

## Parameters

## LPI2C Master Driver

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

Returns

The LPI2C Master Receive Data Register address.

**29.4.5.15** `static void LPI2C_MasterSetWatermarks ( LPI2C_Type * base, size_t txWords, size_t rxWords ) [inline], [static]`

Parameters

|                |                                                                                                                                                                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                                                                                          |
| <i>txWords</i> | Transmit FIFO watermark value in words. The <a href="#">kLPI2C_MasterTxReadyFlag</a> flag is set whenever the number of words in the transmit FIFO is equal or less than <i>txWords</i> . Writing a value equal or greater than the FIFO size is truncated. |
| <i>rxWords</i> | Receive FIFO watermark value in words. The <a href="#">kLPI2C_MasterRxReadyFlag</a> flag is set whenever the number of words in the receive FIFO is greater than <i>rxWords</i> . Writing a value equal or greater than the FIFO size is truncated.         |

**29.4.5.16** `static void LPI2C_MasterGetFifoCounts ( LPI2C_Type * base, size_t * rxCount, size_t * txCount ) [inline], [static]`

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | The LPI2C peripheral base address.                                                                                           |
| out | <i>txCount</i> | Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.  |

**29.4.5.17** `void LPI2C_MasterSetBaudRate ( LPI2C_Type * base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz )`

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

**Note**

Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

## LPI2C Master Driver

### Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.         |
| <i>sourceClock_Hz</i> | LPI2C functional clock frequency in Hertz. |
| <i>baudRate_Hz</i>    | Requested bus frequency in Hertz.          |

### 29.4.5.18 `static bool LPI2C_MasterGetBusIdleState ( LPI2C_Type * base ) [inline], [static]`

Requires the master mode to be enabled.

### Parameters

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

### Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

### 29.4.5.19 `status_t LPI2C_MasterStart ( LPI2C_Type * base, uint8_t address, lpi2c_direction_t dir )`

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

### Parameters

|                |                                                                                                                                                                               |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                            |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                    |
| <i>dir</i>     | Master transfer direction, either <code>kLPI2C_Read</code> or <code>kLPI2C_Write</code> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                                 |                                                                           |
|---------------------------------|---------------------------------------------------------------------------|
| <code>#kStatus_Success</code>   | START signal and address were successfully enqueued in the transmit FIFO. |
| <code>kStatus_LPI2C_Busy</code> | Another master is currently utilizing the bus.                            |

#### 29.4.5.20 `static status_t LPI2C_MasterRepeatedStart ( LPI2C_Type * base, uint8_t address, lpi2c_direction_t dir ) [inline], [static]`

This function is used to send a Repeated START signal when a transfer is already in progress. Like `LPI2C_MasterStart()`, it also sends the specified 7-bit address.

Note

This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

|                      |                                                                                                                                                                               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>base</code>    | The LPI2C peripheral base address.                                                                                                                                            |
| <code>address</code> | 7-bit slave device address, in bits [6:0].                                                                                                                                    |
| <code>dir</code>     | Master transfer direction, either <code>kLPI2C_Read</code> or <code>kLPI2C_Write</code> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                                 |                                                                                    |
|---------------------------------|------------------------------------------------------------------------------------|
| <code>#kStatus_Success</code>   | Repeated START signal and address were successfully enqueued in the transmit FIFO. |
| <code>kStatus_LPI2C_Busy</code> | Another master is currently utilizing the bus.                                     |

#### 29.4.5.21 `status_t LPI2C_MasterSend ( LPI2C_Type * base, const void * txBuff, size_t txSize )`

Sends up to `txSize` number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns `kStatus_LPI2C_Nak`.

## LPI2C Master Driver

### Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                 |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

### Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>#kStatus_Success</i>              | Data was sent successfully.                        |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or over run.                        |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

### 29.4.5.22 `status_t LPI2C_MasterReceive ( LPI2C_Type * base, void * rxBuff, size_t rxSize )`

### Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                 |
| <i>rxBuff</i> | The pointer to the data to be transferred.         |
| <i>rxSize</i> | The length in bytes of the data to be transferred. |

### Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>#kStatus_Success</i>              | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                         |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |



|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_LPI2C_PinLow-Timeout</i> | SCL or SDA were held low longer than the timeout. |
|-------------------------------------|---------------------------------------------------|

#### 29.4.5.23 status\_t LPI2C\_MasterStop ( LPI2C\_Type \* base )

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

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

Return values

|                                      |                                                                                  |
|--------------------------------------|----------------------------------------------------------------------------------|
| <i>#kStatus_Success</i>              | The STOP signal was successfully sent on the bus and the transaction terminated. |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.                                   |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte.                               |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                                                       |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                                                          |
| <i>kStatus_LPI2C_PinLow-Timeout</i>  | SCL or SDA were held low longer than the timeout.                                |

#### 29.4.5.24 status\_t LPI2C\_MasterTransferBlocking ( LPI2C\_Type \* base, lpi2c\_master\_transfer\_t \* transfer )

Note

The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address. |
| <i>transfer</i> | Pointer to the transfer structure. |

Return values

## LPI2C Master Driver

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>#kStatus_Success</i>              | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                         |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

**29.4.5.25** `void LPI2C_MasterTransferCreateHandle ( LPI2C_Type * base,  
lpi2c_master_handle_t * handle, lpi2c_master_transfer_callback_t callback,  
void * userData )`

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_MasterTransferAbort\(\)](#) API shall be called.

Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C master driver handle.                   |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

**29.4.5.26** `status_t LPI2C_MasterTransferNonBlocking ( LPI2C_Type * base,  
lpi2c_master_handle_t * handle, lpi2c_master_transfer_t * transfer )`

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

Return values

|                           |                                                                                                             |
|---------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>#kStatus_Success</i>   | The transaction was started successfully.                                                                   |
| <i>kStatus_LPI2C_Busy</i> | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |

#### 29.4.5.27 **status\_t LPI2C\_MasterTransferGetCount ( LPI2C\_Type \* *base*, lpi2c\_master\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                  |
|     | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

Return values

|                                      |                                                                |
|--------------------------------------|----------------------------------------------------------------|
| <i>#kStatus_Success</i>              |                                                                |
| <i>#kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

#### 29.4.5.28 **void LPI2C\_MasterTransferAbort ( LPI2C\_Type \* *base*, lpi2c\_master\_handle\_t \* *handle* )**

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

## LPI2C Master Driver

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

Return values

|                           |                                                                |
|---------------------------|----------------------------------------------------------------|
| <i>#kStatus_Success</i>   | A transaction was successfully aborted.                        |
| <i>kStatus_LPI2C_Idle</i> | There is not a non-blocking transaction currently in progress. |

**29.4.5.29 void LPI2C\_MasterTransferHandleIRQ ( LPI2C\_Type \* *base*,  
lpi2c\_master\_handle\_t \* *handle* )**

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

## 29.5 LPI2C Slave Driver

### 29.5.1 Overview

#### Data Structures

- struct `lpi2c_slave_config_t`  
*Structure with settings to initialize the LPI2C slave module. [More...](#)*
- struct `lpi2c_slave_transfer_t`  
*LPI2C slave transfer structure. [More...](#)*
- struct `lpi2c_slave_handle_t`  
*LPI2C slave handle structure. [More...](#)*

#### Typedefs

- typedef `void(* lpi2c_slave_transfer_callback_t )(LPI2C_Type *base, lpi2c_slave_transfer_t *transfer, void *userData)`  
*Slave event callback function pointer type.*

#### Enumerations

- enum `_lpi2c_slave_flags` {  
`kLPI2C_SlaveTxReadyFlag = LPI2C_SSR_TDF_MASK,`  
`kLPI2C_SlaveRxReadyFlag = LPI2C_SSR_RDF_MASK,`  
`kLPI2C_SlaveAddressValidFlag = LPI2C_SSR_AVF_MASK,`  
`kLPI2C_SlaveTransmitAckFlag = LPI2C_SSR_TAF_MASK,`  
`kLPI2C_SlaveRepeatedStartDetectFlag = LPI2C_SSR_RSF_MASK,`  
`kLPI2C_SlaveStopDetectFlag = LPI2C_SSR_SDF_MASK,`  
`kLPI2C_SlaveBitErrFlag = LPI2C_SSR_BEF_MASK,`  
`kLPI2C_SlaveFifoErrFlag = LPI2C_SSR_FEF_MASK,`  
`kLPI2C_SlaveAddressMatch0Flag = LPI2C_SSR_AM0F_MASK,`  
`kLPI2C_SlaveAddressMatch1Flag = LPI2C_SSR_AM1F_MASK,`  
`kLPI2C_SlaveGeneralCallFlag = LPI2C_SSR_GCF_MASK,`  
`kLPI2C_SlaveBusyFlag = LPI2C_SSR_SBF_MASK,`  
`kLPI2C_SlaveBusBusyFlag = LPI2C_SSR_BBF_MASK }`  
*LPI2C slave peripheral flags.*
- enum `lpi2c_slave_address_match_t` {  
`kLPI2C_MatchAddress0 = 0U,`  
`kLPI2C_MatchAddress0OrAddress1 = 2U,`  
`kLPI2C_MatchAddress0ThroughAddress1 = 6U }`  
*LPI2C slave address match options.*
- enum `lpi2c_slave_transfer_event_t` {

## LPI2C Slave Driver

```
kLPI2C_SlaveAddressMatchEvent = 0x01U,  
kLPI2C_SlaveTransmitEvent = 0x02U,  
kLPI2C_SlaveReceiveEvent = 0x04U,  
kLPI2C_SlaveTransmitAckEvent = 0x08U,  
kLPI2C_SlaveRepeatedStartEvent = 0x10U,  
kLPI2C_SlaveCompletionEvent = 0x20U,  
kLPI2C_SlaveAllEvents }
```

*Set of events sent to the callback for non blocking slave transfers.*

## Slave initialization and deinitialization

- void **LPI2C\_SlaveGetDefaultConfig** (**lpi2c\_slave\_config\_t** \*slaveConfig)  
*Provides a default configuration for the LPI2C slave peripheral.*
- void **LPI2C\_SlaveInit** (**LPI2C\_Type** \*base, const **lpi2c\_slave\_config\_t** \*slaveConfig, **uint32\_t** sourceClock\_Hz)  
*Initializes the LPI2C slave peripheral.*
- void **LPI2C\_SlaveDeinit** (**LPI2C\_Type** \*base)  
*Deinitializes the LPI2C slave peripheral.*
- static void **LPI2C\_SlaveReset** (**LPI2C\_Type** \*base)  
*Performs a software reset of the LPI2C slave peripheral.*
- static void **LPI2C\_SlaveEnable** (**LPI2C\_Type** \*base, **bool** enable)  
*Enables or disables the LPI2C module as slave.*

## Slave status

- static **uint32\_t** **LPI2C\_SlaveGetStatusFlags** (**LPI2C\_Type** \*base)  
*Gets the LPI2C slave status flags.*
- static void **LPI2C\_SlaveClearStatusFlags** (**LPI2C\_Type** \*base, **uint32\_t** statusMask)  
*Clears the LPI2C status flag state.*

## Slave interrupts

- static void **LPI2C\_SlaveEnableInterrupts** (**LPI2C\_Type** \*base, **uint32\_t** interruptMask)  
*Enables the LPI2C slave interrupt requests.*
- static void **LPI2C\_SlaveDisableInterrupts** (**LPI2C\_Type** \*base, **uint32\_t** interruptMask)  
*Disables the LPI2C slave interrupt requests.*
- static **uint32\_t** **LPI2C\_SlaveGetEnabledInterrupts** (**LPI2C\_Type** \*base)  
*Returns the set of currently enabled LPI2C slave interrupt requests.*

## Slave DMA control

- static void **LPI2C\_SlaveEnableDMA** (**LPI2C\_Type** \*base, **bool** enableAddressValid, **bool** enableRx, **bool** enableTx)  
*Enables or disables the LPI2C slave peripheral DMA requests.*

## Slave bus operations

- static bool [LPI2C\\_SlaveGetBusIdleState](#) (LPI2C\_Type \*base)  
*Returns whether the bus is idle.*
- static void [LPI2C\\_SlaveTransmitAck](#) (LPI2C\_Type \*base, bool ackOrNack)  
*Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.*
- static uint32\_t [LPI2C\\_SlaveGetReceivedAddress](#) (LPI2C\_Type \*base)  
*Returns the slave address sent by the I2C master.*
- status\_t [LPI2C\\_SlaveSend](#) (LPI2C\_Type \*base, const void \*txBuff, size\_t txSize, size\_t \*actualTx-Size)  
*Performs a polling send transfer on the I2C bus.*
- status\_t [LPI2C\\_SlaveReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize, size\_t \*actualRx-Size)  
*Performs a polling receive transfer on the I2C bus.*

## Slave non-blocking

- void [LPI2C\\_SlaveTransferCreateHandle](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, [lpi2c-\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the LPI2C slave non-blocking APIs.*
- status\_t [LPI2C\\_SlaveTransferNonBlocking](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- status\_t [LPI2C\\_SlaveTransferGetCount](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer status during a non-blocking transfer.*
- void [LPI2C\\_SlaveTransferAbort](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Aborts the slave non-blocking transfers.*

## Slave IRQ handler

- void [LPI2C\\_SlaveTransferHandleIRQ](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Reusable routine to handle slave interrupts.*

## 29.5.2 Data Structure Documentation

### 29.5.2.1 struct lpi2c\_slave\_config\_t

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_SlaveGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## LPI2C Slave Driver

### Data Fields

- bool `enableSlave`  
*Enable slave mode.*
- uint8\_t `address0`  
*Slave's 7-bit address.*
- uint8\_t `address1`  
*Alternate slave 7-bit address.*
- `lpi2c_slave_address_match_t` `addressMatchMode`  
*Address matching options.*
- bool `filterDozeEnable`  
*Enable digital glitch filter in doze mode.*
- bool `filterEnable`  
*Enable digital glitch filter.*
- bool `enableGeneralCall`  
*Enable general call address matching.*
- bool `ignoreAck`  
*Continue transfers after a NACK is detected.*
- bool `enableReceivedAddressRead`  
*Enable reading the address received address as the first byte of data.*
- uint32\_t `sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SDA signal.*
- uint32\_t `sclGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SCL signal.*
- uint32\_t `dataValidDelay_ns`  
*Width in nanoseconds of the data valid delay.*
- uint32\_t `clockHoldTime_ns`  
*Width in nanoseconds of the clock hold time.*
- bool `enableAck`  
*Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted.*
- bool `enableTx`  
*Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.*
- bool `enableRx`  
*Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.*
- bool `enableAddress`  
*Enables SCL clock stretching when the address valid flag is asserted.*



### 29.5.2.1.0.13 Field Documentation

29.5.2.1.0.13.1 `bool lpi2c_slave_config_t::enableSlave`

29.5.2.1.0.13.2 `uint8_t lpi2c_slave_config_t::address0`

29.5.2.1.0.13.3 `uint8_t lpi2c_slave_config_t::address1`

29.5.2.1.0.13.4 `lpi2c_slave_address_match_t lpi2c_slave_config_t::addressMatchMode`

29.5.2.1.0.13.5 `bool lpi2c_slave_config_t::filterDozeEnable`

29.5.2.1.0.13.6 `bool lpi2c_slave_config_t::filterEnable`

29.5.2.1.0.13.7 `bool lpi2c_slave_config_t::enableGeneralCall`

29.5.2.1.0.13.8 `bool lpi2c_slave_config_t::enableAck`

Clock stretching occurs when transmitting the 9th bit. When `enableAckSCLStall` is enabled, there is no need to set either `enableRxDDataSCLStall` or `enableAddressSCLStall`.

29.5.2.1.0.13.9 `bool lpi2c_slave_config_t::enableTx`

29.5.2.1.0.13.10 `bool lpi2c_slave_config_t::enableRx`

29.5.2.1.0.13.11 `bool lpi2c_slave_config_t::enableAddress`

29.5.2.1.0.13.12 `bool lpi2c_slave_config_t::ignoreAck`

29.5.2.1.0.13.13 `bool lpi2c_slave_config_t::enableReceivedAddressRead`

29.5.2.1.0.13.14 `uint32_t lpi2c_slave_config_t::sdaGlitchFilterWidth_ns`

29.5.2.1.0.13.15 `uint32_t lpi2c_slave_config_t::sclGlitchFilterWidth_ns`

29.5.2.1.0.13.16 `uint32_t lpi2c_slave_config_t::dataValidDelay_ns`

29.5.2.1.0.13.17 `uint32_t lpi2c_slave_config_t::clockHoldTime_ns`

### 29.5.2.2 `struct lpi2c_slave_transfer_t`

#### Data Fields

- `lpi2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*
- `uint8_t receivedAddress`  
*Matching address send by master.*
- `uint8_t * data`  
*Transfer buffer.*
- `size_t dataSize`  
*Transfer size.*

## LPI2C Slave Driver

- `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.*

### 29.5.2.2.0.14 Field Documentation

#### 29.5.2.2.0.14.1 `lpi2c_slave_transfer_event_t lpi2c_slave_transfer_t::event`

#### 29.5.2.2.0.14.2 `uint8_t lpi2c_slave_transfer_t::receivedAddress`

#### 29.5.2.2.0.14.3 `status_t lpi2c_slave_transfer_t::completionStatus`

Only applies for `kLPI2C_SlaveCompletionEvent`.

#### 29.5.2.2.0.14.4 `size_t lpi2c_slave_transfer_t::transferredCount`

### 29.5.2.3 `struct _lpi2c_slave_handle`

Note

The contents of this structure are private and subject to change.

### Data Fields

- `lpi2c_slave_transfer_t transfer`  
*LPI2C slave transfer copy.*
- `bool isBusy`  
*Whether transfer is busy.*
- `bool wasTransmit`  
*Whether the last transfer was a transmit.*
- `uint32_t eventMask`  
*Mask of enabled events.*
- `uint32_t transferredCount`  
*Count of bytes transferred.*
- `lpi2c_slave_transfer_callback_t callback`  
*Callback function called at transfer event.*
- `void * userData`  
*Callback parameter passed to callback.*

### 29.5.2.3.0.15 Field Documentation

29.5.2.3.0.15.1 `lpi2c_slave_transfer_t lpi2c_slave_handle_t::transfer`

29.5.2.3.0.15.2 `bool lpi2c_slave_handle_t::isBusy`

29.5.2.3.0.15.3 `bool lpi2c_slave_handle_t::wasTransmit`

29.5.2.3.0.15.4 `uint32_t lpi2c_slave_handle_t::eventMask`

29.5.2.3.0.15.5 `uint32_t lpi2c_slave_handle_t::transferredCount`

29.5.2.3.0.15.6 `lpi2c_slave_transfer_callback_t lpi2c_slave_handle_t::callback`

29.5.2.3.0.15.7 `void* lpi2c_slave_handle_t::userData`

### 29.5.3 Typedef Documentation

29.5.3.1 `typedef void(* lpi2c_slave_transfer_callback_t)(LPI2C_Type *base,  
lpi2c_slave_transfer_t *transfer, void *userData)`

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `LPI2C_SlaveSetCallback()` function after you have created a handle.

## LPI2C Slave Driver

Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for the LPI2C instance on which the event occurred.                     |
| <i>transfer</i> | Pointer to transfer descriptor containing values passed to and/or from the callback. |
| <i>userData</i> | Arbitrary pointer-sized value passed from the application.                           |

### 29.5.4 Enumeration Type Documentation

#### 29.5.4.1 enum `_lpi2c_slave_flags`

The following status register flags can be cleared:

- `kLPI2C_SlaveRepeatedStartDetectFlag`
- `kLPI2C_SlaveStopDetectFlag`
- `kLPI2C_SlaveBitErrFlag`
- `kLPI2C_SlaveFifoErrFlag`

All flags except `kLPI2C_SlaveBusyFlag` and `kLPI2C_SlaveBusBusyFlag` can be enabled as interrupts.

Note

These enumerations are meant to be OR'd together to form a bit mask.

Enumerator

- `kLPI2C_SlaveTxReadyFlag` Transmit data flag.
- `kLPI2C_SlaveRxReadyFlag` Receive data flag.
- `kLPI2C_SlaveAddressValidFlag` Address valid flag.
- `kLPI2C_SlaveTransmitAckFlag` Transmit ACK flag.
- `kLPI2C_SlaveRepeatedStartDetectFlag` Repeated start detect flag.
- `kLPI2C_SlaveStopDetectFlag` Stop detect flag.
- `kLPI2C_SlaveBitErrFlag` Bit error flag.
- `kLPI2C_SlaveFifoErrFlag` FIFO error flag.
- `kLPI2C_SlaveAddressMatch0Flag` Address match 0 flag.
- `kLPI2C_SlaveAddressMatch1Flag` Address match 1 flag.
- `kLPI2C_SlaveGeneralCallFlag` General call flag.
- `kLPI2C_SlaveBusyFlag` Master busy flag.
- `kLPI2C_SlaveBusBusyFlag` Bus busy flag.

#### 29.5.4.2 enum `lpi2c_slave_address_match_t`

Enumerator

- `kLPI2C_MatchAddress0` Match only address 0.

***kLPI2C\_MatchAddress0OrAddress1*** Match either address 0 or address 1.

***kLPI2C\_MatchAddress0ThroughAddress1*** Match a range of slave addresses from address 0 through address 1.

### 29.5.4.3 enum lpi2c\_slave\_transfer\_event\_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `LPI2C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

***kLPI2C\_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.

***kLPI2C\_SlaveTransmitEvent*** Callback is requested to provide data to transmit (slave-transmitter role).

***kLPI2C\_SlaveReceiveEvent*** Callback is requested to provide a buffer in which to place received data (slave-receiver role).

***kLPI2C\_SlaveTransmitAckEvent*** Callback needs to either transmit an ACK or NACK.

***kLPI2C\_SlaveRepeatedStartEvent*** A repeated start was detected.

***kLPI2C\_SlaveCompletionEvent*** A stop was detected, completing the transfer.

***kLPI2C\_SlaveAllEvents*** Bit mask of all available events.

## 29.5.5 Function Documentation

### 29.5.5.1 void LPI2C\_SlaveGetDefaultConfig ( lpi2c\_slave\_config\_t \* slaveConfig )

This function provides the following default configuration for the LPI2C slave peripheral:

```
* slaveConfig->enableSlave           = true;
* slaveConfig->address0              = 0U;
* slaveConfig->address1              = 0U;
* slaveConfig->addressMatchMode      = kLPI2C_MatchAddress0;
* slaveConfig->filterDozeEnable      = true;
* slaveConfig->filterEnable          = true;
* slaveConfig->enableGeneralCall     = false;
* slaveConfig->sclStall.enableAck    = false;
* slaveConfig->sclStall.enableTx     = true;
* slaveConfig->sclStall.enableRx     = true;
* slaveConfig->sclStall.enableAddress = true;
* slaveConfig->ignoreAck             = false;
* slaveConfig->enableReceivedAddressRead = false;
* slaveConfig->sdaGlitchFilterWidth_ns = 0;
* slaveConfig->sclGlitchFilterWidth_ns = 0;
* slaveConfig->dataValidDelay_ns    = 0;
* slaveConfig->clockHoldTime_ns     = 0;
*
```

## LPI2C Slave Driver

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with `LPI2C_SlaveInit()`. Be sure to override at least the `address0` member of the configuration structure with the desired slave address.

Parameters

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

### 29.5.5.2 void LPI2C\_SlaveInit ( LPI2C\_Type \* *base*, const lpi2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *sourceClock\_Hz* )

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

|                             |                                                                                                                                        |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <code>base</code>           | The LPI2C peripheral base address.                                                                                                     |
| <code>slaveConfig</code>    | User provided peripheral configuration. Use <code>LPI2C_SlaveGetDefaultConfig()</code> to get a set of defaults that you can override. |
| <code>sourceClock_Hz</code> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.          |

### 29.5.5.3 void LPI2C\_SlaveDeinit ( LPI2C\_Type \* *base* )

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|                   |                                    |
|-------------------|------------------------------------|
| <code>base</code> | The LPI2C peripheral base address. |
|-------------------|------------------------------------|

### 29.5.5.4 static void LPI2C\_SlaveReset ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

---

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

**29.5.5.5** `static void LPI2C_SlaveEnable ( LPI2C_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                    |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as slave. |

**29.5.5.6** `static uint32_t LPI2C_SlaveGetStatusFlags ( LPI2C_Type * base ) [inline], [static]`

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

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

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_lpi2c\\_slave\\_flags](#)

**29.5.5.7** `static void LPI2C_SlaveClearStatusFlags ( LPI2C_Type * base, uint32_t statusMask ) [inline], [static]`

The following status register flags can be cleared:

- [kLPI2C\\_SlaveRepeatedStartDetectFlag](#)
- [kLPI2C\\_SlaveStopDetectFlag](#)
- [kLPI2C\\_SlaveBitErrFlag](#)
- [kLPI2C\\_SlaveFifoErrFlag](#)

Attempts to clear other flags has no effect.

## LPI2C Slave Driver

### Parameters

|                   |                                                                                                                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                                  |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_lpi2c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_SlaveGetStatusFlags()</a> . |

### See Also

[\\_lpi2c\\_slave\\_flags](#).

#### 29.5.5.8 **static void LPI2C\_SlaveEnableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

### Parameters

|                      |                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                   |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 29.5.5.9 **static void LPI2C\_SlaveDisableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

### Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 29.5.5.10 **static uint32\_t LPI2C\_SlaveGetEnabledInterrupts ( LPI2C\_Type \* *base* ) [inline], [static]**



## Parameters

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

## Returns

A bitmask composed of [\\_lpi2c\\_slave\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

**29.5.5.11** `static void LPI2C_SlaveEnableDMA ( LPI2C_Type * base, bool enableAddressValid, bool enableRx, bool enableTx ) [inline], [static]`

## Parameters

|                           |                                                                                                                                                                    |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | The LPI2C peripheral base address.                                                                                                                                 |
| <i>enableAddressValid</i> | Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request. |
| <i>enableRx</i>           | Enable flag for the receive data DMA request. Pass true for enable, false for disable.                                                                             |
| <i>enableTx</i>           | Enable flag for the transmit data DMA request. Pass true for enable, false for disable.                                                                            |

**29.5.5.12** `static bool LPI2C_SlaveGetBusIdleState ( LPI2C_Type * base ) [inline], [static]`

Requires the slave mode to be enabled.

## Parameters

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

## Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

**29.5.5.13** `static void LPI2C_SlaveTransmitAck ( LPI2C_Type * base, bool ackOrNack ) [inline], [static]`

Use this function to send an ACK or NAK when the [kLPI2C\\_SlaveTransmitAckFlag](#) is asserted. This only happens if you enable the `sclStall.enableAck` field of the [lpi2c\\_slave\\_config\\_t](#) configuration structure used to initialize the slave peripheral.

## LPI2C Slave Driver

### Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | The LPI2C peripheral base address.       |
| <i>ackOrNack</i> | Pass true for an ACK or false for a NAK. |

### 29.5.5.14 `static uint32_t LPI2C_SlaveGetReceivedAddress ( LPI2C_Type * base ) [inline], [static]`

This function should only be called if the [kLPI2C\\_SlaveAddressValidFlag](#) is asserted.

### Parameters

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

### Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

### 29.5.5.15 `status_t LPI2C_SlaveSend ( LPI2C_Type * base, const void * txBuff, size_t txSize, size_t * actualTxSize )`

### Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C peripheral base address.                 |
|     | <i>txBuff</i>       | The pointer to the data to be transferred.         |
|     | <i>txSize</i>       | The length in bytes of the data to be transferred. |
| out | <i>actualTxSize</i> |                                                    |

### Returns

Error or success status returned by API.

### 29.5.5.16 `status_t LPI2C_SlaveReceive ( LPI2C_Type * base, void * rxBuff, size_t rxSize, size_t * actualRxSize )`

## Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C peripheral base address.                 |
|     | <i>rxBuff</i>       | The pointer to the data to be transferred.         |
|     | <i>rxSize</i>       | The length in bytes of the data to be transferred. |
| out | <i>actualRxSize</i> |                                                    |

## Returns

Error or success status returned by API.

**29.5.5.17 void LPI2C\_SlaveTransferCreateHandle ( LPI2C\_Type \* *base*, lpi2c\_slave\_handle\_t \* *handle*, lpi2c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_SlaveTransferAbort\(\)](#) API shall be called.

## Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

## Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C slave driver handle.                    |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

**29.5.5.18 status\_t LPI2C\_SlaveTransferNonBlocking ( LPI2C\_Type \* *base*, lpi2c\_slave\_handle\_t \* *handle*, uint32\_t *eventMask* )**

Call this API after calling [I2C\\_SlaveInit\(\)](#) and [LPI2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to [LPI2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [lpi2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The

## LPI2C Slave Driver

[kLPI2C\\_SlaveTransmitEvent](#) and [kLPI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kLPI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

### Parameters

|                  |                                                                                                                                                                                                                                                                                                        |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The LPI2C peripheral base address.                                                                                                                                                                                                                                                                     |
| <i>handle</i>    | Pointer to #lpi2c_slave_handle_t structure which stores the transfer state.                                                                                                                                                                                                                            |
| <i>eventMask</i> | Bit mask formed by OR'ing together <a href="#">lpi2c_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="#">kLPI2C_SlaveAllEvents</a> to enable all events. |

### Return values

|                           |                                                           |
|---------------------------|-----------------------------------------------------------|
| <i>#kStatus_Success</i>   | Slave transfers were successfully started.                |
| <i>kStatus_LPI2C_Busy</i> | Slave transfers have already been started on this handle. |

### 29.5.5.19 **status\_t LPI2C\_SlaveTransferGetCount ( LPI2C\_Type \* base, lpi2c\_slave\_handle\_t \* handle, size\_t \* count )**

### Parameters

|     |               |                                                                                                       |
|-----|---------------|-------------------------------------------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                                                    |
|     | <i>handle</i> | Pointer to i2c_slave_handle_t structure.                                                              |
| out | <i>count</i>  | Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required. |

### Return values

|                                      |  |
|--------------------------------------|--|
| <i>#kStatus_Success</i>              |  |
| <i>#kStatus_NoTransferInProgress</i> |  |

### 29.5.5.20 **void LPI2C\_SlaveTransferAbort ( LPI2C\_Type \* base, lpi2c\_slave\_handle\_t \* handle )**

## Note

This API could be called at any time to stop slave for handling the bus events.

## Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                          |
| <i>handle</i> | Pointer to #lpi2c_slave_handle_t structure which stores the transfer state. |

## Return values

|                           |  |
|---------------------------|--|
| <i>#kStatus_Success</i>   |  |
| <i>kStatus_LPI2C_Idle</i> |  |

#### 29.5.5.21 void LPI2C\_SlaveTransferHandleIRQ ( LPI2C\_Type \* *base*, lpi2c\_slave\_handle\_t \* *handle* )

## Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

## Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                          |
| <i>handle</i> | Pointer to #lpi2c_slave_handle_t structure which stores the transfer state. |

## LPI2C Master DMA Driver

### 29.6 LPI2C Master DMA Driver

#### 29.6.1 Overview

#### Data Structures

- struct `lpi2c_master_edma_handle_t`  
*Driver handle for master DMA APIs. [More...](#)*

#### Typedefs

- typedef void(\* `lpi2c_master_edma_transfer_callback_t`)(LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, status\_t completionStatus, void \*userData)  
*Master DMA completion callback function pointer type.*

#### Master DMA

- void `LPI2C_MasterCreateEDMAHandle` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, edma\_handle\_t \*rxDmaHandle, edma\_handle\_t \*txDmaHandle, lpi2c\_master\_edma\_transfer\_callback\_t callback, void \*userData)  
*Create a new handle for the LPI2C master DMA APIs.*
- status\_t `LPI2C_MasterTransferEDMA` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, lpi2c\_master\_transfer\_t \*transfer)  
*Performs a non-blocking DMA-based transaction on the I2C bus.*
- status\_t `LPI2C_MasterTransferGetCountEDMA` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- status\_t `LPI2C_MasterTransferAbortEDMA` (LPI2C\_Type \*base, lpi2c\_master\_edma\_handle\_t \*handle)  
*Terminates a non-blocking LPI2C master transmission early.*

#### 29.6.2 Data Structure Documentation

##### 29.6.2.1 struct `_lpi2c_master_edma_handle`

Note

The contents of this structure are private and subject to change.

#### Data Fields

- LPI2C\_Type \* `base`  
*LPI2C base pointer.*
- bool `isBusy`

- *Transfer state machine current state.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint16_t commandBuffer` [7]  
*LPI2C command sequence.*
- `lpi2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `lpi2c_master_edma_transfer_callback_t completionCallback`  
*Callback function pointer.*
- `void * userData`  
*Application data passed to callback.*
- `edma_handle_t * rx`  
*Handle for receive DMA channel.*
- `edma_handle_t * tx`  
*Handle for transmit DMA channel.*
- `edma_tcd_t tcds` [2]  
*Software TCD.*

#### 29.6.2.1.0.16 Field Documentation

- 29.6.2.1.0.16.1 `LPI2C_Type* lpi2c_master_edma_handle_t::base`
- 29.6.2.1.0.16.2 `bool lpi2c_master_edma_handle_t::isBusy`
- 29.6.2.1.0.16.3 `uint8_t lpi2c_master_edma_handle_t::nbytes`
- 29.6.2.1.0.16.4 `uint16_t lpi2c_master_edma_handle_t::commandBuffer[7]`
- 29.6.2.1.0.16.5 `lpi2c_master_transfer_t lpi2c_master_edma_handle_t::transfer`
- 29.6.2.1.0.16.6 `lpi2c_master_edma_transfer_callback_t lpi2c_master_edma_handle_t::completionCallback`
- 29.6.2.1.0.16.7 `void* lpi2c_master_edma_handle_t::userData`
- 29.6.2.1.0.16.8 `edma_handle_t* lpi2c_master_edma_handle_t::rx`
- 29.6.2.1.0.16.9 `edma_handle_t* lpi2c_master_edma_handle_t::tx`
- 29.6.2.1.0.16.10 `edma_tcd_t lpi2c_master_edma_handle_t::tcds[2]`

Two are allocated to provide enough room to align to 32-bytes.

### 29.6.3 Typedef Documentation

**29.6.3.1** `typedef void(* lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)`

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [LPI2C\\_MasterCreateEDMAHandle\(\)](#).



## Parameters

|                          |                                                                                 |
|--------------------------|---------------------------------------------------------------------------------|
| <i>base</i>              | The LPI2C peripheral base address.                                              |
| <i>handle</i>            | Handle associated with the completed transfer.                                  |
| <i>completion-Status</i> | Either #kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                      |

## 29.6.4 Function Documentation

**29.6.4.1 void LPI2C\_MasterCreateEDMAHandle ( LPI2C\_Type \* *base*, lpi2c\_master\_edma\_handle\_t \* *handle*, edma\_handle\_t \* *rxDmaHandle*, edma\_handle\_t \* *txDmaHandle*, lpi2c\_master\_edma\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_MasterTransferAbortEDMA\(\)](#) API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

## Parameters

|     |                    |                                                                                           |
|-----|--------------------|-------------------------------------------------------------------------------------------|
|     | <i>base</i>        | The LPI2C peripheral base address.                                                        |
| out | <i>handle</i>      | Pointer to the LPI2C master driver handle.                                                |
|     | <i>rxDmaHandle</i> | Handle for the eDMA receive channel. Created by the user prior to calling this function.  |
|     | <i>txDmaHandle</i> | Handle for the eDMA transmit channel. Created by the user prior to calling this function. |
|     | <i>callback</i>    | User provided pointer to the asynchronous callback function.                              |
|     | <i>userData</i>    | User provided pointer to the application callback data.                                   |

**29.6.4.2 status\_t LPI2C\_MasterTransferEDMA ( LPI2C\_Type \* *base*, lpi2c\_master\_edma\_handle\_t \* *handle*, lpi2c\_master\_transfer\_t \* *transfer* )**

The callback specified when the *handle* was created is invoked when the transaction has completed.

## LPI2C Master DMA Driver

### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

### Return values

|                           |                                                                                                          |
|---------------------------|----------------------------------------------------------------------------------------------------------|
| <i>#kStatus_Success</i>   | The transaction was started successfully.                                                                |
| <i>kStatus_LPI2C_Busy</i> | Either another master is currently utilizing the bus, or another DMA transaction is already in progress. |

### 29.6.4.3 **status\_t LPI2C\_MasterTransferGetCountEDMA ( LPI2C\_Type \* *base*, lpi2c\_master\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

#### Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                  |
|     | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

#### Return values

|                                      |                                                       |
|--------------------------------------|-------------------------------------------------------|
| <i>#kStatus_Success</i>              |                                                       |
| <i>#kStatus_NoTransferInProgress</i> | There is not a DMA transaction currently in progress. |

### 29.6.4.4 **status\_t LPI2C\_MasterTransferAbortEDMA ( LPI2C\_Type \* *base*, lpi2c\_master\_edma\_handle\_t \* *handle* )**

#### Note

It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

## Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

## Return values

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>#kStatus_Success</i>   | A transaction was successfully aborted.               |
| <i>kStatus_LPI2C_Idle</i> | There is not a DMA transaction currently in progress. |

## LPI2C FreeRTOS Driver

### 29.7 LPI2C FreeRTOS Driver

#### 29.7.1 Overview

##### Driver version

- #define **FSL\_LPI2C\_FREERTOS\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 5))  
*LPI2C freertos driver version 2.1.5.*

##### LPI2C RTOS Operation

- status\_t **LPI2C\_RTOS\_Init** (lpi2c\_rtos\_handle\_t \*handle, LPI2C\_Type \*base, const lpi2c\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes LPI2C.*
- status\_t **LPI2C\_RTOS\_Deinit** (lpi2c\_rtos\_handle\_t \*handle)  
*Deinitializes the LPI2C.*
- status\_t **LPI2C\_RTOS\_Transfer** (lpi2c\_rtos\_handle\_t \*handle, lpi2c\_master\_transfer\_t \*transfer)  
*Performs I2C transfer.*

#### 29.7.2 Macro Definition Documentation

##### 29.7.2.1 #define FSL\_LPI2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 5))

#### 29.7.3 Function Documentation

##### 29.7.3.1 status\_t LPI2C\_RTOS\_Init ( lpi2c\_rtos\_handle\_t \* handle, LPI2C\_Type \* base, const lpi2c\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )

This function initializes the LPI2C module and related RTOS context.

Parameters

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <i>handle</i>       | The RTOS LPI2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the LPI2C instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up LPI2C in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the LPI2C module.                              |

Returns

status of the operation.

**29.7.3.2 status\_t LPI2C\_RTOS\_Deinit ( lpi2c\_rtos\_handle\_t \* *handle* )**

This function deinitializes the LPI2C module and related RTOS context.

## LPI2C FreeRTOS Driver

### Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPI2C handle. |
|---------------|------------------------|

### 29.7.3.3 `status_t LPI2C_RTOS_Transfer ( lpi2c_rtos_handle_t * handle, lpi2c_master_transfer_t * transfer )`

This function performs an I2C transfer using LPI2C module according to data given in the transfer structure.

### Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS LPI2C handle.                        |
| <i>transfer</i> | Structure specifying the transfer parameters. |

### Returns

status of the operation.



## Chapter 30

# LPSPI: Low Power Serial Peripheral Interface

### 30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power Serial Peripheral Interface (LPSPI) module of MCUXpresso SDK devices.

#### Modules

- [LPSPI FreeRTOS Driver](#)
- [LPSPI Peripheral driver](#)
- [LPSPI eDMA Driver](#)

## **LPSPI Peripheral driver**

### **30.2 LPSPI Peripheral driver**

#### **30.2.1 Overview**

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

#### **30.2.2 Function groups**

##### **30.2.2.1 LPSPI Initialization and De-initialization**

This function group initializes the default configuration structure for master and slave, initializes the LPSPI master with a master configuration, initializes the LPSPI slave with a slave configuration, and de-initializes the LPSPI module.

##### **30.2.2.2 LPSPI Basic Operation**

This function group enables/disables the LPSPI module both interrupt and DMA, gets the data register address for the DMA transfer, sets master and slave, starts and stops the transfer, and so on.

##### **30.2.2.3 LPSPI Transfer Operation**

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

##### **30.2.2.4 LPSPI Status Operation**

This function group gets/clears the LPSPI status.

##### **30.2.2.5 LPSPI Block Transfer Operation**

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

#### **30.2.3 Typical use case**

##### **30.2.3.1 Master Operation**

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



### 30.2.3.2 Slave Operation

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

#### Data Structures

- struct `lpspi_master_config_t`  
*LPSPI master configuration structure. [More...](#)*
- struct `lpspi_slave_config_t`  
*LPSPI slave configuration structure. [More...](#)*
- struct `lpspi_transfer_t`  
*LPSPI master/slave transfer structure. [More...](#)*
- struct `lpspi_master_handle_t`  
*LPSPI master transfer handle structure used for transactional API. [More...](#)*
- struct `lpspi_slave_handle_t`  
*LPSPI slave transfer handle structure used for transactional API. [More...](#)*

#### Macros

- #define `LPSPI_DUMMY_DATA` (0x00U)  
*LPSPI dummy data if no Tx data.*
- #define `LPSPI_MASTER_PCS_SHIFT` (4U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_MASTER_PCS_MASK` (0xF0U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_SLAVE_PCS_SHIFT` (4U)  
*LPSPI slave PCS shift macro , internal used.*
- #define `LPSPI_SLAVE_PCS_MASK` (0xF0U)  
*LPSPI slave PCS shift macro , internal used.*

#### Typedefs

- typedef void(\* `lpspi_master_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, status\_t status, void \*userData)  
*Master completion callback function pointer type.*
- typedef void(\* `lpspi_slave_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, status\_t status, void \*userData)  
*Slave completion callback function pointer type.*

#### Enumerations

- enum `_lpspi_status` {  
`kStatus_LPSPI_Busy` = MAKE\_STATUS(kStatusGroup\_LPSPI, 0),  
`kStatus_LPSPI_Error` = MAKE\_STATUS(kStatusGroup\_LPSPI, 1),  
`kStatus_LPSPI_Idle` = MAKE\_STATUS(kStatusGroup\_LPSPI, 2),

## LPSPI Peripheral driver

```
kStatus_LPSPI_OutOfRange = MAKE_STATUS(kStatusGroup_LPSPI, 3) }
```

*Status for the LPSPI driver.*

- enum `_lpspi_flags` {  
    `kLPSPI_TxDataRequestFlag` = `LPSPI_SR_TDF_MASK`,  
    `kLPSPI_RxDataReadyFlag` = `LPSPI_SR_RDF_MASK`,  
    `kLPSPI_WordCompleteFlag` = `LPSPI_SR_WCF_MASK`,  
    `kLPSPI_FrameCompleteFlag` = `LPSPI_SR_FCF_MASK`,  
    `kLPSPI_TransferCompleteFlag` = `LPSPI_SR_TCF_MASK`,  
    `kLPSPI_TransmitErrorFlag` = `LPSPI_SR_TEF_MASK`,  
    `kLPSPI_ReceiveErrorFlag` = `LPSPI_SR_REF_MASK`,  
    `kLPSPI_DataMatchFlag` = `LPSPI_SR_DMF_MASK`,  
    `kLPSPI_ModuleBusyFlag` = `LPSPI_SR_MBF_MASK`,  
    `kLPSPI_AllStatusFlag` }  
*LPSPI status flags in SPIx\_SR register.*
- enum `_lpspi_interrupt_enable` {  
    `kLPSPI_TxInterruptEnable` = `LPSPI_IER_TDIE_MASK`,  
    `kLPSPI_RxInterruptEnable` = `LPSPI_IER_RDIE_MASK`,  
    `kLPSPI_WordCompleteInterruptEnable` = `LPSPI_IER_WCIE_MASK`,  
    `kLPSPI_FrameCompleteInterruptEnable` = `LPSPI_IER_FCIE_MASK`,  
    `kLPSPI_TransferCompleteInterruptEnable` = `LPSPI_IER_TCIE_MASK`,  
    `kLPSPI_TransmitErrorInterruptEnable` = `LPSPI_IER_TEIE_MASK`,  
    `kLPSPI_ReceiveErrorInterruptEnable` = `LPSPI_IER_REIE_MASK`,  
    `kLPSPI_DataMatchInterruptEnable` = `LPSPI_IER_DMIE_MASK`,  
    `kLPSPI_AllInterruptEnable` }  
*LPSPI interrupt source.*
- enum `_lpspi_dma_enable` {  
    `kLPSPI_TxDmaEnable` = `LPSPI_DER_TDDE_MASK`,  
    `kLPSPI_RxDmaEnable` = `LPSPI_DER_RDDE_MASK` }  
*LPSPI DMA source.*
- enum `lpspi_master_slave_mode_t` {  
    `kLPSPI_Master` = 1U,  
    `kLPSPI_Slave` = 0U }  
*LPSPI master or slave mode configuration.*
- enum `lpspi_which_pcs_t` {  
    `kLPSPI_Pcs0` = 0U,  
    `kLPSPI_Pcs1` = 1U,  
    `kLPSPI_Pcs2` = 2U,  
    `kLPSPI_Pcs3` = 3U }  
*LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).*
- enum `lpspi_pcs_polarity_config_t` {  
    `kLPSPI_PcsActiveHigh` = 1U,  
    `kLPSPI_PcsActiveLow` = 0U }  
*LPSPI Peripheral Chip Select (PCS) Polarity configuration.*
- enum `_lpspi_pcs_polarity` {

- `kLPSPI_Pcs0ActiveLow = 1U << 0,`
- `kLPSPI_Pcs1ActiveLow = 1U << 1,`
- `kLPSPI_Pcs2ActiveLow = 1U << 2,`
- `kLPSPI_Pcs3ActiveLow = 1U << 3,`
- `kLPSPI_PcsAllActiveLow = 0xFU }`
- LPSPI Peripheral Chip Select (PCS) Polarity.*
- enum `lpspi_clock_polarity_t` {
  - `kLPSPI_ClockPolarityActiveHigh = 0U,`
  - `kLPSPI_ClockPolarityActiveLow = 1U }`*LPSPI clock polarity configuration.*
- enum `lpspi_clock_phase_t` {
  - `kLPSPI_ClockPhaseFirstEdge = 0U,`
  - `kLPSPI_ClockPhaseSecondEdge = 1U }`*LPSPI clock phase configuration.*
- enum `lpspi_shift_direction_t` {
  - `kLPSPI_MsbFirst = 0U,`
  - `kLPSPI_LsbFirst = 1U }`*LPSPI data shifter direction options.*
- enum `lpspi_host_request_select_t` {
  - `kLPSPI_HostReqExtPin = 0U,`
  - `kLPSPI_HostReqInternalTrigger = 1U }`*LPSPI Host Request select configuration.*
- enum `lpspi_match_config_t` {
  - `kLPSI_MatchDisabled = 0x0U,`
  - `kLPSI_1stWordEqualsM0orM1 = 0x2U,`
  - `kLPSI_AnyWordEqualsM0orM1 = 0x3U,`
  - `kLPSI_1stWordEqualsM0and2ndWordEqualsM1 = 0x4U,`
  - `kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 = 0x5U,`
  - `kLPSI_1stWordAndM1EqualsM0andM1 = 0x6U,`
  - `kLPSI_AnyWordAndM1EqualsM0andM1 = 0x7U }`*LPSPI Match configuration options.*
- enum `lpspi_pin_config_t` {
  - `kLPSPI_SdiInSdoOut = 0U,`
  - `kLPSPI_SdiInSdiOut = 1U,`
  - `kLPSPI_SdoInSdoOut = 2U,`
  - `kLPSPI_SdoInSdiOut = 3U }`*LPSPI pin (SDO and SDI) configuration.*
- enum `lpspi_data_out_config_t` {
  - `kLpspiDataOutRetained = 0U,`
  - `kLpspiDataOutTristate = 1U }`*LPSPI data output configuration.*
- enum `lpspi_transfer_width_t` {
  - `kLPSPI_SingleBitXfer = 0U,`
  - `kLPSPI_TwoBitXfer = 1U,`
  - `kLPSPI_FourBitXfer = 2U }`*LPSPI transfer width configuration.*
- enum `lpspi_delay_type_t` {

## LPSPI Peripheral driver

```
kLPSPI_PcsToSck = 1U,  
kLPSPI_LastSckToPcs,  
kLPSPI_BetweenTransfer }
```

*LPSPI delay type selection.*

- enum `_lpspi_transfer_config_flag_for_master` {  
kLPSPI\_MasterPcs0 = 0U << LPSPI\_MASTER\_PCS\_SHIFT,  
kLPSPI\_MasterPcs1 = 1U << LPSPI\_MASTER\_PCS\_SHIFT,  
kLPSPI\_MasterPcs2 = 2U << LPSPI\_MASTER\_PCS\_SHIFT,  
kLPSPI\_MasterPcs3 = 3U << LPSPI\_MASTER\_PCS\_SHIFT,  
kLPSPI\_MasterPcsContinuous = 1U << 20,  
kLPSPI\_MasterByteSwap }

*Use this enumeration for LPSPI master transfer configFlags.*

- enum `_lpspi_transfer_config_flag_for_slave` {  
kLPSPI\_SlavePcs0 = 0U << LPSPI\_SLAVE\_PCS\_SHIFT,  
kLPSPI\_SlavePcs1 = 1U << LPSPI\_SLAVE\_PCS\_SHIFT,  
kLPSPI\_SlavePcs2 = 2U << LPSPI\_SLAVE\_PCS\_SHIFT,  
kLPSPI\_SlavePcs3 = 3U << LPSPI\_SLAVE\_PCS\_SHIFT,  
kLPSPI\_SlaveByteSwap }

*Use this enumeration for LPSPI slave transfer configFlags.*

- enum `_lpspi_transfer_state` {  
kLPSPI\_Idle = 0x0U,  
kLPSPI\_Busy,  
kLPSPI\_Error }

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

## Variables

- volatile uint8\_t `g_lpspiDummyData` []  
*Global variable for dummy data value setting.*

## Driver version

- #define `FSL_LPSPI_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*LPSPI driver version 2.0.1.*

## Initialization and deinitialization

- void `LPSPI_MasterInit` (LPSPI\_Type \*base, const `lpspi_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the LPSPI master.*
- void `LPSPI_MasterGetDefaultConfig` (`lpspi_master_config_t` \*masterConfig)  
*Sets the `lpspi_master_config_t` structure to default values.*
- void `LPSPI_SlaveInit` (LPSPI\_Type \*base, const `lpspi_slave_config_t` \*slaveConfig)  
*LPSPI slave configuration.*

- void [LPSPI\\_SlaveGetDefaultConfig](#) ([lpspi\\_slave\\_config\\_t](#) \*slaveConfig)  
*Sets the [lpspi\\_slave\\_config\\_t](#) structure to default values.*
- void [LPSPI\\_Deinit](#) ([LPSPI\\_Type](#) \*base)  
*De-initializes the LPSPI peripheral.*
- void [LPSPI\\_Reset](#) ([LPSPI\\_Type](#) \*base)  
*Restores the LPSPI peripheral to reset state.*
- static void [LPSPI\\_Enable](#) ([LPSPI\\_Type](#) \*base, bool enable)  
*Enables the LPSPI peripheral and sets the MCR MDIS to 0.*

### Status

- static [uint32\\_t](#) [LPSPI\\_GetStatusFlags](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI status flag state.*
- static [uint32\\_t](#) [LPSPI\\_GetTxFifoSize](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Tx FIFO size.*
- static [uint32\\_t](#) [LPSPI\\_GetRxFifoSize](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Rx FIFO size.*
- static [uint32\\_t](#) [LPSPI\\_GetTxFifoCount](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Tx FIFO count.*
- static [uint32\\_t](#) [LPSPI\\_GetRxFifoCount](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Rx FIFO count.*
- static void [LPSPI\\_ClearStatusFlags](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) statusFlags)  
*Clears the LPSPI status flag.*

### Interrupts

- static void [LPSPI\\_EnableInterrupts](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Enables the LPSPI interrupts.*
- static void [LPSPI\\_DisableInterrupts](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Disables the LPSPI interrupts.*

### DMA Control

- static void [LPSPI\\_EnableDMA](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Enables the LPSPI DMA request.*
- static void [LPSPI\\_DisableDMA](#) ([LPSPI\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Disables the LPSPI DMA request.*
- static [uint32\\_t](#) [LPSPI\\_GetTxRegisterAddress](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Transmit Data Register address for a DMA operation.*
- static [uint32\\_t](#) [LPSPI\\_GetRxRegisterAddress](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Receive Data Register address for a DMA operation.*

### Bus Operations

- [uint32\\_t](#) [LPSPI\\_GetInstance](#) ([LPSPI\\_Type](#) \*base)  
*Get instance number for LPSPI module.*

## LPSPi Peripheral driver

- bool [LPSPi\\_CheckTransferArgument](#) ([lpspi\\_transfer\\_t](#) \*transfer, uint32\_t bitsPerFrame, uint32\_t bytesPerFrame)  
*Check the argument for transfer.*
- static void [LPSPi\\_SetMasterSlaveMode](#) (LPSPi\_Type \*base, [lpspi\\_master\\_slave\\_mode\\_t](#) mode)  
*Configures the LPSPi for either master or slave.*
- static bool [LPSPi\\_IsMaster](#) (LPSPi\_Type \*base)  
*Returns whether the LPSPi module is in master mode.*
- static void [LPSPi\\_FlushFifo](#) (LPSPi\_Type \*base, bool flushTxFifo, bool flushRxFifo)  
*Flushes the LPSPi FIFOs.*
- static void [LPSPi\\_SetFifoWatermarks](#) (LPSPi\_Type \*base, uint32\_t txWater, uint32\_t rxWater)  
*Sets the transmit and receive FIFO watermark values.*
- static void [LPSPi\\_SetAllPcsPolarity](#) (LPSPi\_Type \*base, uint32\_t mask)  
*Configures all LPSPi peripheral chip select polarities simultaneously.*
- static void [LPSPi\\_SetFrameSize](#) (LPSPi\_Type \*base, uint32\_t frameSize)  
*Configures the frame size.*
- uint32\_t [LPSPi\\_MasterSetBaudRate](#) (LPSPi\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz, uint32\_t \*tcrPrescaleValue)  
*Sets the LPSPi baud rate in bits per second.*
- void [LPSPi\\_MasterSetDelayScaler](#) (LPSPi\_Type \*base, uint32\_t scaler, [lpspi\\_delay\\_type\\_t](#) whichDelay)  
*Manually configures a specific LPSPi delay parameter (module must be disabled to change the delay values).*
- uint32\_t [LPSPi\\_MasterSetDelayTimes](#) (LPSPi\_Type \*base, uint32\_t delayTimeInNanoSec, [lpspi\\_delay\\_type\\_t](#) whichDelay, uint32\_t srcClock\_Hz)  
*Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).*
- static void [LPSPi\\_WriteData](#) (LPSPi\_Type \*base, uint32\_t data)  
*Writes data into the transmit data buffer.*
- static uint32\_t [LPSPi\\_ReadData](#) (LPSPi\_Type \*base)  
*Reads data from the data buffer.*
- void [LPSPi\\_SetDummyData](#) (LPSPi\_Type \*base, uint8\_t dummyData)  
*Set up the dummy data.*

## Transactional

- void [LPSPi\\_MasterTransferCreateHandle](#) (LPSPi\_Type \*base, [lpspi\\_master\\_handle\\_t](#) \*handle, [lpspi\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the LPSPi master handle.*
- status\_t [LPSPi\\_MasterTransferBlocking](#) (LPSPi\_Type \*base, [lpspi\\_transfer\\_t](#) \*transfer)  
*LPSPi master transfer data using a polling method.*
- status\_t [LPSPi\\_MasterTransferNonBlocking](#) (LPSPi\_Type \*base, [lpspi\\_master\\_handle\\_t](#) \*handle, [lpspi\\_transfer\\_t](#) \*transfer)  
*LPSPi master transfer data using an interrupt method.*
- status\_t [LPSPi\\_MasterTransferGetCount](#) (LPSPi\_Type \*base, [lpspi\\_master\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the master transfer remaining bytes.*
- void [LPSPi\\_MasterTransferAbort](#) (LPSPi\_Type \*base, [lpspi\\_master\\_handle\\_t](#) \*handle)  
*LPSPi master abort transfer which uses an interrupt method.*
- void [LPSPi\\_MasterTransferHandleIRQ](#) (LPSPi\_Type \*base, [lpspi\\_master\\_handle\\_t](#) \*handle)

- LPSPI Master IRQ handler function.*

• void [LPSPI\\_SlaveTransferCreateHandle](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, [lpspi\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)

*Initializes the LPSPI slave handle.*
- status\_t [LPSPI\\_SlaveTransferNonBlocking](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, [lpspi\\_transfer\\_t](#) \*transfer)

*LPSPI slave transfer data using an interrupt method.*
- status\_t [LPSPI\\_SlaveTransferGetCount](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, size\_t \*count)

*Gets the slave transfer remaining bytes.*
- void [LPSPI\\_SlaveTransferAbort](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle)

*LPSPI slave aborts a transfer which uses an interrupt method.*
- void [LPSPI\\_SlaveTransferHandleIRQ](#) (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle)

*LPSPI Slave IRQ handler function.*

## 30.2.4 Data Structure Documentation

### 30.2.4.1 struct lpspi\_master\_config\_t

#### Data Fields

- uint32\_t [baudRate](#)  
*Baud Rate for LPSPI.*
- uint32\_t [bitsPerFrame](#)  
*Bits per frame, minimum 8, maximum 4096.*
- [lpspi\\_clock\\_polarity\\_t](#) [cpol](#)  
*Clock polarity.*
- [lpspi\\_clock\\_phase\\_t](#) [cpha](#)  
*Clock phase.*
- [lpspi\\_shift\\_direction\\_t](#) [direction](#)  
*MSB or LSB data shift direction.*
- uint32\_t [pcsToSckDelayInNanoSec](#)  
*PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay.*
- uint32\_t [lastSckToPcsDelayInNanoSec](#)  
*Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay.*
- uint32\_t [betweenTransferDelayInNanoSec](#)  
*After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay.*
- [lpspi\\_which\\_pcs\\_t](#) [whichPcs](#)  
*Desired Peripheral Chip Select (PCS).*
- [lpspi\\_pcs\\_polarity\\_config\\_t](#) [pcsActiveHighOrLow](#)  
*Desired PCS active high or low.*
- [lpspi\\_pin\\_config\\_t](#) [pinCfg](#)  
*Configures which pins are used for input and output data during single bit transfers.*
- [lpspi\\_data\\_out\\_config\\_t](#) [dataOutConfig](#)  
*Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).*

## LPSPI Peripheral driver

### 30.2.4.1.0.17 Field Documentation

**30.2.4.1.0.17.1** `uint32_t lpspi_master_config_t::baudRate`

**30.2.4.1.0.17.2** `uint32_t lpspi_master_config_t::bitsPerFrame`

**30.2.4.1.0.17.3** `lpspi_clock_polarity_t lpspi_master_config_t::cpol`

**30.2.4.1.0.17.4** `lpspi_clock_phase_t lpspi_master_config_t::cpha`

**30.2.4.1.0.17.5** `lpspi_shift_direction_t lpspi_master_config_t::direction`

**30.2.4.1.0.17.6** `uint32_t lpspi_master_config_t::pcsToSckDelayInNanoSec`

It sets the boundary value if out of range.

**30.2.4.1.0.17.7** `uint32_t lpspi_master_config_t::lastSckToPcsDelayInNanoSec`

It sets the boundary value if out of range.

**30.2.4.1.0.17.8** `uint32_t lpspi_master_config_t::betweenTransferDelayInNanoSec`

It sets the boundary value if out of range.

**30.2.4.1.0.17.9** `lpspi_which_pcs_t lpspi_master_config_t::whichPcs`

**30.2.4.1.0.17.10** `lpspi_pin_config_t lpspi_master_config_t::pinCfg`

**30.2.4.1.0.17.11** `lpspi_data_out_config_t lpspi_master_config_t::dataOutConfig`

### 30.2.4.2 struct `lpspi_slave_config_t`

#### Data Fields

- `uint32_t bitsPerFrame`  
*Bits per frame, minimum 8, maximum 4096.*
- `lpspi_clock_polarity_t cpol`  
*Clock polarity.*
- `lpspi_clock_phase_t cpha`  
*Clock phase.*
- `lpspi_shift_direction_t direction`  
*MSB or LSB data shift direction.*
- `lpspi_which_pcs_t whichPcs`  
*Desired Peripheral Chip Select (pcs)*
- `lpspi_pcs_polarity_config_t pcsActiveHighOrLow`  
*Desired PCS active high or low.*
- `lpspi_pin_config_t pinCfg`  
*Configures which pins are used for input and output data during single bit transfers.*
- `lpspi_data_out_config_t dataOutConfig`  
*Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).*



**30.2.4.2.0.18 Field Documentation****30.2.4.2.0.18.1** `uint32_t lpspi_slave_config_t::bitsPerFrame`**30.2.4.2.0.18.2** `lpspi_clock_polarity_t lpspi_slave_config_t::cpol`**30.2.4.2.0.18.3** `lpspi_clock_phase_t lpspi_slave_config_t::cpha`**30.2.4.2.0.18.4** `lpspi_shift_direction_t lpspi_slave_config_t::direction`**30.2.4.2.0.18.5** `lpspi_pin_config_t lpspi_slave_config_t::pinCfg`**30.2.4.2.0.18.6** `lpspi_data_out_config_t lpspi_slave_config_t::dataOutConfig`**30.2.4.3 struct lpspi\_transfer\_t****Data Fields**

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- volatile `size_t dataSize`  
*Transfer bytes.*
- `uint32_t configFlags`  
*Transfer transfer configuration flags.*

**30.2.4.3.0.19 Field Documentation****30.2.4.3.0.19.1** `uint8_t* lpspi_transfer_t::txData`**30.2.4.3.0.19.2** `uint8_t* lpspi_transfer_t::rxData`**30.2.4.3.0.19.3** volatile `size_t lpspi_transfer_t::dataSize`**30.2.4.3.0.19.4** `uint32_t lpspi_transfer_t::configFlags`

Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

**30.2.4.4 struct \_lpspi\_master\_handle**

Forward declaration of the `_lpspi_master_handle` typedefs.

**Data Fields**

- volatile `bool isPcsContinuous`  
*Is PCS continuous in transfer.*
- volatile `bool writeTcrInIsr`

## LPSPI Peripheral driver

- *A flag that whether should write TCR in ISR.*  
volatile bool **isByteSwap**
- *A flag that whether should byte swap.*  
volatile uint8\_t **fifoSize**  
*FIFO dataSize.*
- volatile uint8\_t **rxWatermark**  
*Rx watermark.*
- volatile uint8\_t **bytesEachWrite**  
*Bytes for each write TDR.*
- volatile uint8\_t **bytesEachRead**  
*Bytes for each read RDR.*
- uint8\_t \*volatile **txData**  
*Send buffer.*
- uint8\_t \*volatile **rxData**  
*Receive buffer.*
- volatile size\_t **txRemainingByteCount**  
*Number of bytes remaining to send.*
- volatile size\_t **rxRemainingByteCount**  
*Number of bytes remaining to receive.*
- volatile uint32\_t **writeRegRemainingTimes**  
*Write TDR register remaining times.*
- volatile uint32\_t **readRegRemainingTimes**  
*Read RDR register remaining times.*
- uint32\_t **totalByteCount**  
*Number of transfer bytes.*
- uint32\_t **txBuffIfNull**  
*Used if the txData is NULL.*
- volatile uint8\_t **state**  
*LPSPI transfer state , `_lpspi_transfer_state`.*
- **lpspi\_master\_transfer\_callback\_t** **callback**  
*Completion callback.*
- void \* **userData**  
*Callback user data.*

**30.2.4.4.0.20 Field Documentation**

- 30.2.4.4.0.20.1** `volatile bool lpspi_master_handle_t::isPcsContinuous`
- 30.2.4.4.0.20.2** `volatile bool lpspi_master_handle_t::writeTcrInlSr`
- 30.2.4.4.0.20.3** `volatile bool lpspi_master_handle_t::isByteSwap`
- 30.2.4.4.0.20.4** `volatile uint8_t lpspi_master_handle_t::fifoSize`
- 30.2.4.4.0.20.5** `volatile uint8_t lpspi_master_handle_t::rxWatermark`
- 30.2.4.4.0.20.6** `volatile uint8_t lpspi_master_handle_t::bytesEachWrite`
- 30.2.4.4.0.20.7** `volatile uint8_t lpspi_master_handle_t::bytesEachRead`
- 30.2.4.4.0.20.8** `uint8_t* volatile lpspi_master_handle_t::txData`
- 30.2.4.4.0.20.9** `uint8_t* volatile lpspi_master_handle_t::rxData`
- 30.2.4.4.0.20.10** `volatile size_t lpspi_master_handle_t::txRemainingByteCount`
- 30.2.4.4.0.20.11** `volatile size_t lpspi_master_handle_t::rxRemainingByteCount`
- 30.2.4.4.0.20.12** `volatile uint32_t lpspi_master_handle_t::writeRegRemainingTimes`
- 30.2.4.4.0.20.13** `volatile uint32_t lpspi_master_handle_t::readRegRemainingTimes`
- 30.2.4.4.0.20.14** `uint32_t lpspi_master_handle_t::txBuffIfNull`
- 30.2.4.4.0.20.15** `volatile uint8_t lpspi_master_handle_t::state`
- 30.2.4.4.0.20.16** `lpspi_master_transfer_callback_t lpspi_master_handle_t::callback`
- 30.2.4.4.0.20.17** `void* lpspi_master_handle_t::userData`

**30.2.4.5 struct `_lpspi_slave_handle`**

Forward declaration of the `_lpspi_slave_handle` typedefs.

**Data Fields**

- volatile bool `isByteSwap`  
*A flag that whether should byte swap.*
- volatile uint8\_t `fifoSize`  
*FIFO dataSize.*
- volatile uint8\_t `rxWatermark`  
*Rx watermark.*
- volatile uint8\_t `bytesEachWrite`  
*Bytes for each write TDR.*

## LPSPI Peripheral driver

- volatile uint8\_t **bytesEachRead**  
*Bytes for each read RDR.*
- uint8\_t \*volatile **txData**  
*Send buffer.*
- uint8\_t \*volatile **rxData**  
*Receive buffer.*
- volatile size\_t **txRemainingByteCount**  
*Number of bytes remaining to send.*
- volatile size\_t **rxRemainingByteCount**  
*Number of bytes remaining to receive.*
- volatile uint32\_t **writeRegRemainingTimes**  
*Write TDR register remaining times.*
- volatile uint32\_t **readRegRemainingTimes**  
*Read RDR register remaining times.*
- uint32\_t **totalByteCount**  
*Number of transfer bytes.*
- volatile uint8\_t **state**  
*LPSPI transfer state , `_lpspi_transfer_state`.*
- volatile uint32\_t **errorCount**  
*Error count for slave transfer.*
- **lpspi\_slave\_transfer\_callback\_t** **callback**  
*Completion callback.*
- void \* **userData**  
*Callback user data.*

**30.2.4.5.0.21 Field Documentation**

- 30.2.4.5.0.21.1** `volatile bool lpspi_slave_handle_t::isByteSwap`
- 30.2.4.5.0.21.2** `volatile uint8_t lpspi_slave_handle_t::fifoSize`
- 30.2.4.5.0.21.3** `volatile uint8_t lpspi_slave_handle_t::rxWatermark`
- 30.2.4.5.0.21.4** `volatile uint8_t lpspi_slave_handle_t::bytesEachWrite`
- 30.2.4.5.0.21.5** `volatile uint8_t lpspi_slave_handle_t::bytesEachRead`
- 30.2.4.5.0.21.6** `uint8_t* volatile lpspi_slave_handle_t::txData`
- 30.2.4.5.0.21.7** `uint8_t* volatile lpspi_slave_handle_t::rxData`
- 30.2.4.5.0.21.8** `volatile size_t lpspi_slave_handle_t::txRemainingByteCount`
- 30.2.4.5.0.21.9** `volatile size_t lpspi_slave_handle_t::rxRemainingByteCount`
- 30.2.4.5.0.21.10** `volatile uint32_t lpspi_slave_handle_t::writeRegRemainingTimes`
- 30.2.4.5.0.21.11** `volatile uint32_t lpspi_slave_handle_t::readRegRemainingTimes`
- 30.2.4.5.0.21.12** `volatile uint8_t lpspi_slave_handle_t::state`
- 30.2.4.5.0.21.13** `volatile uint32_t lpspi_slave_handle_t::errorCount`
- 30.2.4.5.0.21.14** `lpspi_slave_transfer_callback_t lpspi_slave_handle_t::callback`
- 30.2.4.5.0.21.15** `void* lpspi_slave_handle_t::userData`

**30.2.5 Macro Definition Documentation**

**30.2.5.1** `#define FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

**30.2.5.2** `#define LPSPI_DUMMY_DATA (0x00U)`

Dummy data used for tx if there is not txData.

## LPSPI Peripheral driver

**30.2.5.3** `#define LPSPI_MASTER_PCS_SHIFT (4U)`

**30.2.5.4** `#define LPSPI_MASTER_PCS_MASK (0xF0U)`

**30.2.5.5** `#define LPSPI_SLAVE_PCS_SHIFT (4U)`

**30.2.5.6** `#define LPSPI_SLAVE_PCS_MASK (0xF0U)`

### 30.2.6 Typedef Documentation

**30.2.6.1** `typedef void(* lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)`

## Parameters

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

### 30.2.6.2 typedef void(\* lpspi\_slave\_transfer\_callback\_t)(LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, status\_t status, void \*userData)

## Parameters

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

## 30.2.7 Enumeration Type Documentation

### 30.2.7.1 enum \_lpspi\_status

## Enumerator

- kStatus\_LPSPI\_Busy* LPSPI transfer is busy.
- kStatus\_LPSPI\_Error* LPSPI driver error.
- kStatus\_LPSPI\_Idle* LPSPI is idle.
- kStatus\_LPSPI\_OutOfRange* LPSPI transfer out Of range.

### 30.2.7.2 enum \_lpspi\_flags

## Enumerator

- kLPSPI\_TxDataRequestFlag* Transmit data flag.
- kLPSPI\_RxDataReadyFlag* Receive data flag.
- kLPSPI\_WordCompleteFlag* Word Complete flag.
- kLPSPI\_FrameCompleteFlag* Frame Complete flag.
- kLPSPI\_TransferCompleteFlag* Transfer Complete flag.
- kLPSPI\_TransmitErrorFlag* Transmit Error flag (FIFO underrun)
- kLPSPI\_ReceiveErrorFlag* Receive Error flag (FIFO overrun)
- kLPSPI\_DataMatchFlag* Data Match flag.

## LPSPI Peripheral driver

*kLPSPI\_ModuleBusyFlag* Module Busy flag.

*kLPSPI\_AllStatusFlag* Used for clearing all w1c status flags.

### 30.2.7.3 enum `_lpspi_interrupt_enable`

Enumerator

*kLPSPI\_TxInterruptEnable* Transmit data interrupt enable.

*kLPSPI\_RxInterruptEnable* Receive data interrupt enable.

*kLPSPI\_WordCompleteInterruptEnable* Word complete interrupt enable.

*kLPSPI\_FrameCompleteInterruptEnable* Frame complete interrupt enable.

*kLPSPI\_TransferCompleteInterruptEnable* Transfer complete interrupt enable.

*kLPSPI\_TransmitErrorInterruptEnable* Transmit error interrupt enable(FIFO underrun)

*kLPSPI\_ReceiveErrorInterruptEnable* Receive Error interrupt enable (FIFO overrun)

*kLPSPI\_DataMatchInterruptEnable* Data Match interrupt enable.

*kLPSPI\_AllInterruptEnable* All above interrupts enable.

### 30.2.7.4 enum `_lpspi_dma_enable`

Enumerator

*kLPSPI\_TxDmaEnable* Transmit data DMA enable.

*kLPSPI\_RxDmaEnable* Receive data DMA enable.

### 30.2.7.5 enum `lpspi_master_slave_mode_t`

Enumerator

*kLPSPI\_Master* LPSPI peripheral operates in master mode.

*kLPSPI\_Slave* LPSPI peripheral operates in slave mode.

### 30.2.7.6 enum `lpspi_which_pcs_t`

Enumerator

*kLPSPI\_Pcs0* PCS[0].

*kLPSPI\_Pcs1* PCS[1].

*kLPSPI\_Pcs2* PCS[2].

*kLPSPI\_Pcs3* PCS[3].



**30.2.7.7 enum lpspi\_pcs\_polarity\_config\_t**

Enumerator

*kLPSPI\_PcsActiveHigh* PCS Active High (idles low)*kLPSPI\_PcsActiveLow* PCS Active Low (idles high)**30.2.7.8 enum \_lpspi\_pcs\_polarity**

Enumerator

*kLPSPI\_Pcs0ActiveLow* Pcs0 Active Low (idles high).*kLPSPI\_Pcs1ActiveLow* Pcs1 Active Low (idles high).*kLPSPI\_Pcs2ActiveLow* Pcs2 Active Low (idles high).*kLPSPI\_Pcs3ActiveLow* Pcs3 Active Low (idles high).*kLPSPI\_PcsAllActiveLow* Pcs0 to Pcs5 Active Low (idles high).**30.2.7.9 enum lpspi\_clock\_polarity\_t**

Enumerator

*kLPSPI\_ClockPolarityActiveHigh* CPOL=0. Active-high LPSPI clock (idles low)*kLPSPI\_ClockPolarityActiveLow* CPOL=1. Active-low LPSPI clock (idles high)**30.2.7.10 enum lpspi\_clock\_phase\_t**

Enumerator

*kLPSPI\_ClockPhaseFirstEdge* CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.*kLPSPI\_ClockPhaseSecondEdge* CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.**30.2.7.11 enum lpspi\_shift\_direction\_t**

Enumerator

*kLPSPI\_MsbFirst* Data transfers start with most significant bit.*kLPSPI\_LsbFirst* Data transfers start with least significant bit.

## LPSPi Peripheral driver

### 30.2.7.12 enum `lpspi_host_request_select_t`

Enumerator

*kLPSPI\_HostReqExtPin* Host Request is an ext pin.

*kLPSPI\_HostReqInternalTrigger* Host Request is an internal trigger.

### 30.2.7.13 enum `lpspi_match_config_t`

Enumerator

*kLPSI\_MatchDisabled* LPSPi Match Disabled.

*kLPSI\_1stWordEqualsM0orM1* LPSPi Match Enabled.

*kLPSI\_AnyWordEqualsM0orM1* LPSPi Match Enabled.

*kLPSI\_1stWordEqualsM0and2ndWordEqualsM1* LPSPi Match Enabled.

*kLPSI\_AnyWordEqualsM0andNxtWordEqualsM1* LPSPi Match Enabled.

*kLPSI\_1stWordAndM1EqualsM0andM1* LPSPi Match Enabled.

*kLPSI\_AnyWordAndM1EqualsM0andM1* LPSPi Match Enabled.

### 30.2.7.14 enum `lpspi_pin_config_t`

Enumerator

*kLPSPI\_SdiInSdoOut* LPSPi SDI input, SDO output.

*kLPSPI\_SdiInSdiOut* LPSPi SDI input, SDI output.

*kLPSPI\_SdoInSdoOut* LPSPi SDO input, SDO output.

*kLPSPI\_SdoInSdiOut* LPSPi SDO input, SDI output.

### 30.2.7.15 enum `lpspi_data_out_config_t`

Enumerator

*kLpspiDataOutRetained* Data out retains last value when chip select is de-asserted.

*kLpspiDataOutTristate* Data out is tristated when chip select is de-asserted.

### 30.2.7.16 enum `lpspi_transfer_width_t`

Enumerator

*kLPSPI\_SingleBitXfer* 1-bit shift at a time, data out on SDO, in on SDI (normal mode)

*kLPSPI\_TwoBitXfer* 2-bits shift out on SDO/SDI and in on SDO/SDI

*kLPSPI\_FourBitXfer* 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

**30.2.7.17 enum lpspi\_delay\_type\_t**

Enumerator

- kLPSPI\_PcsToSck* PCS-to-SCK delay.
- kLPSPI\_LastSckToPcs* Last SCK edge to PCS delay.
- kLPSPI\_BetweenTransfer* Delay between transfers.

**30.2.7.18 enum \_lpspi\_transfer\_config\_flag\_for\_master**

Enumerator

- kLPSPI\_MasterPcs0* LPSPI master transfer use PCS0 signal.
- kLPSPI\_MasterPcs1* LPSPI master transfer use PCS1 signal.
- kLPSPI\_MasterPcs2* LPSPI master transfer use PCS2 signal.
- kLPSPI\_MasterPcs3* LPSPI master transfer use PCS3 signal.
- kLPSPI\_MasterPcsContinuous* Is PCS signal continuous.
- kLPSPI\_MasterByteSwap* Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi\_shift\_direction\_t to MSB).
  1. If you set bitPerFrame = 8 , no matter the kLPSPI\_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
  2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_MasterByteSwap flag.
  3. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_MasterByteSwap flag.

**30.2.7.19 enum \_lpspi\_transfer\_config\_flag\_for\_slave**

Enumerator

- kLPSPI\_SlavePcs0* LPSPI slave transfer use PCS0 signal.
- kLPSPI\_SlavePcs1* LPSPI slave transfer use PCS1 signal.
- kLPSPI\_SlavePcs2* LPSPI slave transfer use PCS2 signal.
- kLPSPI\_SlavePcs3* LPSPI slave transfer use PCS3 signal.
- kLPSPI\_SlaveByteSwap* Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi\_shift\_direction\_t to MSB).
  1. If you set bitPerFrame = 8 , no matter the kLPSPI\_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
  2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_SlaveByteSwap flag.

## LPSPi Peripheral driver

3. If you set `bitPerFrame = 32` : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the `kLPSPi_SlaveByteSwap` flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the `kLPSPi_SlaveByteSwap` flag.

### 30.2.7.20 enum \_lpspi\_transfer\_state

Enumerator

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

## 30.2.8 Function Documentation

### 30.2.8.1 void LPSPi\_MasterInit ( LPSPi\_Type \* base, const lpspi\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )

Parameters

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

### 30.2.8.2 void LPSPi\_MasterGetDefaultConfig ( lpspi\_master\_config\_t \* masterConfig )

This API initializes the configuration structure for [LPSPi\\_MasterInit\(\)](#). The initialized structure can remain unchanged in [LPSPi\\_MasterInit\(\)](#), or can be modified before calling the [LPSPi\\_MasterInit\(\)](#). Example:

```
* lpspi_master_config_t masterConfig;  
* LPSPi_MasterGetDefaultConfig(&masterConfig);  
*
```

Parameters

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| <i>masterConfig</i> | pointer to <a href="#">lpspi_master_config_t</a> structure |
|---------------------|------------------------------------------------------------|

### 30.2.8.3 void LPSPi\_SlaveInit ( LPSPi\_Type \* base, const lpspi\_slave\_config\_t \* slaveConfig )

Parameters

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

#### 30.2.8.4 void LPSPI\_SlaveGetDefaultConfig ( lpspi\_slave\_config\_t \* *slaveConfig* )

This API initializes the configuration structure for [LPSPI\\_SlaveInit\(\)](#). The initialized structure can remain unchanged in [LPSPI\\_SlaveInit\(\)](#) or can be modified before calling the [LPSPI\\_SlaveInit\(\)](#). Example:

```
* lpspi_slave_config_t slaveConfig;
* LPSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

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

#### 30.2.8.5 void LPSPI\_Deinit ( LPSPI\_Type \* *base* )

Call this API to disable the LPSPI clock.

Parameters

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

#### 30.2.8.6 void LPSPI\_Reset ( LPSPI\_Type \* *base* )

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

Parameters

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

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

## LPSPI Peripheral driver

### Parameters

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

**30.2.8.8** `static uint32_t LPSPI_GetStatusFlags ( LPSPI_Type * base ) [inline], [static]`

### Parameters

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

### Returns

The LPSPI status(in SR register).

**30.2.8.9** `static uint32_t LPSPI_GetTxFifoSize ( LPSPI_Type * base ) [inline], [static]`

### Parameters

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

### Returns

The LPSPI Tx FIFO size.

**30.2.8.10** `static uint32_t LPSPI_GetRxFifoSize ( LPSPI_Type * base ) [inline], [static]`

### Parameters

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

### Returns

The LPSPI Rx FIFO size.

**30.2.8.11** `static uint32_t LPSPI_GetTxFifoCount ( LPSPI_Type * base ) [inline], [static]`

Parameters

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

Returns

The number of words in the transmit FIFO.

**30.2.8.12 static uint32\_t LPSPI\_GetRxFifoCount ( LPSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The number of words in the receive FIFO.

**30.2.8.13 static void LPSPI\_ClearStatusFlags ( LPSPI\_Type \* *base*, uint32\_t *statusFlags* ) [inline], [static]**

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the `_lpspi_flags`. Example usage:

```
* LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag |
    kLPSPI_RxDataReadyFlag);
*
```

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | LPSPI peripheral address.                                  |
| <i>statusFlags</i> | The status flag used from type <code>_lpspi_flags</code> . |

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

**30.2.8.14 static void LPSPI\_EnableInterrupts ( LPSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
* LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable |
    kLPSPI_RxInterruptEnable );
*
```

## LPSPI Peripheral driver

### Parameters

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                               |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_interrupt_enable</code> . |

### 30.2.8.15 `static void LPSPI_DisableInterrupts ( LPSPI_Type * base, uint32_t mask )` `[inline], [static]`

```
* LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable |  
    kLPSPI_RxInterruptEnable );  
*
```

### Parameters

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                               |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_interrupt_enable</code> . |

### 30.2.8.16 `static void LPSPI_EnableDMA ( LPSPI_Type * base, uint32_t mask )` `[inline], [static]`

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

```
* LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable |  
    kLPSPI_RxDmaEnable);  
*
```

### Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                         |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_dma_enable</code> . |

### 30.2.8.17 `static void LPSPI_DisableDMA ( LPSPI_Type * base, uint32_t mask )` `[inline], [static]`

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

```
* LPSPI_DisableDMA(base, kLPSPI_TxDmaEnable |  
    kLPSPI_RxDmaEnable);  
*
```



Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                         |
| <i>mask</i> | The interrupt mask; Use the enum <code>_lpspi_dma_enable</code> . |

### 30.2.8.18 `static uint32_t LPSPI_GetTxRegisterAddress ( LPSPI_Type * base )` `[inline], [static]`

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

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

Returns

The LPSPI Transmit Data Register address.

### 30.2.8.19 `static uint32_t LPSPI_GetRxRegisterAddress ( LPSPI_Type * base )` `[inline], [static]`

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

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

Returns

The LPSPI Receive Data Register address.

### 30.2.8.20 `uint32_t LPSPI_GetInstance ( LPSPI_Type * base )`

Parameters

---

## LPSPI Peripheral driver

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

Returns

Return the value of LPSPI instance.

### 30.2.8.21 **bool LPSPI\_CheckTransferArgument ( lpspi\_transfer\_t \* *transfer*, uint32\_t *bitsPerFrame*, uint32\_t *bytesPerFrame* )**

Parameters

|                     |                                 |
|---------------------|---------------------------------|
| <i>transfer</i>     | the transfer struct to be used. |
| <i>bitPerFrame</i>  | The bit size of one frame.      |
| <i>bytePerFrame</i> | The byte size of one frame.     |

Returns

Return true for right and false for wrong.

### 30.2.8.22 **static void LPSPI\_SetMasterSlaveMode ( LPSPI\_Type \* *base*, lpspi\_master\_slave\_mode\_t *mode* ) [inline], [static]**

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

Parameters

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

### 30.2.8.23 **static bool LPSPI\_IsMaster ( LPSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

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

**30.2.8.24** `static void LPSPI_FlushFifo ( LPSPI_Type * base, bool flushTxFifo, bool flushRxFifo ) [inline], [static]`

## LPSPI Peripheral driver

### Parameters

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

### 30.2.8.25 static void LPSPI\_SetFifoWatermarks ( LPSPI\_Type \* *base*, uint32\_t *txWater*, uint32\_t *rxWater* ) [inline], [static]

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

### Parameters

|                |                                                                                                |
|----------------|------------------------------------------------------------------------------------------------|
| <i>base</i>    | LPSPI peripheral address.                                                                      |
| <i>txWater</i> | The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated. |
| <i>rxWater</i> | The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated. |

### 30.2.8.26 static void LPSPI\_SetAllPcsPolarity ( LPSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
* LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow |  
    kLPSPI_Pcs1ActiveLow);  
*
```

### Parameters

|             |                                                                        |
|-------------|------------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                              |
| <i>mask</i> | The PCS polarity mask; Use the enum <code>_lpspi_pcs_polarity</code> . |

**30.2.8.27 static void LPSPI\_SetFrameSize ( LPSPI\_Type \* *base*, uint32\_t *frameSize* )  
[inline], [static]**

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | LPSPI peripheral address.         |
| <i>frameSize</i> | The frame size in number of bits. |

**30.2.8.28 uint32\_t LPSPI\_MasterSetBaudRate ( LPSPI\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz*, uint32\_t \* *tcrPrescaleValue* )**

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale *tcrPrescaleValue* parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

|                          |                                                   |
|--------------------------|---------------------------------------------------|
| <i>base</i>              | LPSPI peripheral address.                         |
| <i>baudRate_Bps</i>      | The desired baud rate in bits per second.         |
| <i>srcClock_Hz</i>       | Module source input clock in Hertz.               |
| <i>tcrPrescale-Value</i> | The TCR prescale value needed to program the TCR. |

## LPSPI Peripheral driver

### Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

### 30.2.8.29 void LPSPI\_MasterSetDelayScaler ( LPSPI\_Type \* *base*, uint32\_t *scaler*, lpspi\_delay\_type\_t *whichDelay* )

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi\_delay\_type\_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

### Parameters

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| <i>base</i>       | LPSPI peripheral address.                                           |
| <i>scaler</i>     | The 8-bit delay value 0x00 to 0xFF (255).                           |
| <i>whichDelay</i> | The desired delay to configure, must be of type lpspi_delay_type_t. |

### 30.2.8.30 uint32\_t LPSPI\_MasterSetDelayTimes ( LPSPI\_Type \* *base*, uint32\_t *delayTimeInNanoSec*, lpspi\_delay\_type\_t *whichDelay*, uint32\_t *srcClock\_Hz* )

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi\_delay\_type\_t.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the  $\text{delayTime} = \text{LPSPI\_clockSource} / (\text{PRESCALE} * \text{Delay\_scaler})$ .

Parameters

|                           |                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------|
| <i>base</i>               | LPSPI peripheral address.                                                                   |
| <i>delayTimeInNanoSec</i> | The desired delay value in nano-seconds.                                                    |
| <i>whichDelay</i>         | The desired delay to configuration, which must be of type <code>lpspi_delay_type_t</code> . |
| <i>srcClock_Hz</i>        | Module source input clock in Hertz.                                                         |

Returns

actual Calculated delay value in nano-seconds.

**30.2.8.31** `static void LPSPI_WriteData ( LPSPI_Type * base, uint32_t data ) [inline], [static]`

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | LPSPI peripheral address. |
| <i>data</i> | The data word to be sent. |

**30.2.8.32** `static uint32_t LPSPI_ReadData ( LPSPI_Type * base ) [inline], [static]`

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

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

Returns

The data read from the data buffer.

**30.2.8.33** `void LPSPI_SetDummyData ( LPSPI_Type * base, uint8_t dummyData )`

## LPSPI Peripheral driver

### Parameters

|                  |                                                                                                                                                                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | LPSPI peripheral address.                                                                                                                                                                                                                       |
| <i>dummyData</i> | Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated. |

**30.2.8.34 void LPSPI\_MasterTransferCreateHandle ( LPSPI\_Type \* *base*,  
lpspi\_master\_handle\_t \* *handle*, lpspi\_master\_transfer\_callback\_t *callback*,  
void \* *userData* )**

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

### Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                      |
| <i>handle</i>   | LPSPI handle pointer to lpspi_master_handle_t. |
| <i>callback</i> | DSPI callback.                                 |
| <i>userData</i> | callback function parameter.                   |

**30.2.8.35 status\_t LPSPI\_MasterTransferBlocking ( LPSPI\_Type \* *base*, lpspi\_transfer\_t  
\* *transfer* )**

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

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                              |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure. |

### Returns

status of status\_t.



### 30.2.8.36 **status\_t LPSPI\_MasterTransferNonBlocking ( LPSPI\_Type \* *base*, lpspi\_master\_handle\_t \* *handle*, lpspi\_transfer\_t \* *transfer* )**

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                                   |
| <i>handle</i>   | pointer to lpspi_master_handle_t structure which stores the transfer state. |
| <i>transfer</i> | pointer to lpspi_transfer_t structure.                                      |

Returns

status of status\_t.

### 30.2.8.37 **status\_t LPSPI\_MasterTransferGetCount ( LPSPI\_Type \* *base*, lpspi\_master\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the master transfer remaining bytes.

Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                   |
| <i>handle</i> | pointer to lpspi_master_handle_t 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.

### 30.2.8.38 **void LPSPI\_MasterTransferAbort ( LPSPI\_Type \* *base*, lpspi\_master\_handle\_t \* *handle* )**

This function aborts a transfer which uses an interrupt method.

## LPSPI Peripheral driver

Parameters

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

### 30.2.8.39 void LPSPI\_MasterTransferHandleIRQ ( LPSPI\_Type \* *base*, lpspi\_master\_handle\_t \* *handle* )

This function processes the LPSPI transmit and receive IRQ.

Parameters

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

### 30.2.8.40 void LPSPI\_SlaveTransferCreateHandle ( LPSPI\_Type \* *base*, lpspi\_slave\_handle\_t \* *handle*, lpspi\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

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

Parameters

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

### 30.2.8.41 status\_t LPSPI\_SlaveTransferNonBlocking ( LPSPI\_Type \* *base*, lpspi\_slave\_handle\_t \* *handle*, lpspi\_transfer\_t \* *transfer* )

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

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

Returns

status of `status_t`.

**30.2.8.42** `status_t LPSPI_SlaveTransferGetCount ( LPSPI_Type * base, lpspi_slave_handle_t * handle, size_t * count )`

This function gets the slave transfer remaining bytes.

Parameters

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

Returns

status of `status_t`.

**30.2.8.43** `void LPSPI_SlaveTransferAbort ( LPSPI_Type * base, lpspi_slave_handle_t * handle )`

This function aborts a transfer which uses an interrupt method.

Parameters

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

**30.2.8.44** `void LPSPI_SlaveTransferHandleIRQ ( LPSPI_Type * base, lpspi_slave_handle_t * handle )`

This function processes the LPSPi transmit and receives an IRQ.

## LPSPI Peripheral driver

Parameters

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

### 30.2.9 Variable Documentation

#### 30.2.9.1 `volatile uint8_t g_lpspiDummyData[]`

## 30.3 LPSPI eDMA Driver

### 30.3.1 Overview

#### Data Structures

- struct `lpspi_master_edma_handle_t`  
*LPSPI master eDMA transfer handle structure used for transactional API. [More...](#)*
- struct `lpspi_slave_edma_handle_t`  
*LPSPI slave eDMA transfer handle structure used for transactional API. [More...](#)*

#### Typedefs

- typedef `void(* lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)`  
*Completion callback function pointer type.*
- typedef `void(* lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, status_t status, void *userData)`  
*Completion callback function pointer type.*

#### Functions

- void `LPSPI_MasterTransferCreateHandleEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, `lpspi_master_edma_transfer_callback_t` callback, void \*userData, `edma_handle_t` \*edmaRxRegToRxDataHandle, `edma_handle_t` \*edmaTxDataToTxRegHandle)  
*Initializes the LPSPI master eDMA handle.*
- status\_t `LPSPI_MasterTransferEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, `lpspi_transfer_t` \*transfer)  
*LPSPI master transfer data using eDMA.*
- void `LPSPI_MasterTransferAbortEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle)  
*LPSPI master aborts a transfer which is using eDMA.*
- status\_t `LPSPI_MasterTransferGetCountEDMA` (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the master eDMA transfer remaining bytes.*
- void `LPSPI_SlaveTransferCreateHandleEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, `lpspi_slave_edma_transfer_callback_t` callback, void \*userData, `edma_handle_t` \*edmaRxRegToRxDataHandle, `edma_handle_t` \*edmaTxDataToTxRegHandle)  
*Initializes the LPSPI slave eDMA handle.*
- status\_t `LPSPI_SlaveTransferEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, `lpspi_transfer_t` \*transfer)  
*LPSPI slave transfers data using eDMA.*
- void `LPSPI_SlaveTransferAbortEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle)  
*LPSPI slave aborts a transfer which is using eDMA.*
- status\_t `LPSPI_SlaveTransferGetCountEDMA` (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, size\_t \*count)

## LPSPI eDMA Driver

*Gets the slave eDMA transfer remaining bytes.*

### Driver version

- #define `FSL_LPSPI_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*LPSPI EDMA driver version 2.0.2.*

## 30.3.2 Data Structure Documentation

### 30.3.2.1 struct `_lpspi_master_edma_handle`

Forward declaration of the `_lpspi_master_edma_handle` typedefs.

#### Data Fields

- volatile bool `isPcsContinuous`  
*Is PCS continuous in transfer.*
- volatile bool `isByteSwap`  
*A flag that whether should byte swap.*
- volatile uint8\_t `fifoSize`  
*FIFO dataSize.*
- volatile uint8\_t `rxWatermark`  
*Rx watermark.*
- volatile uint8\_t `bytesEachWrite`  
*Bytes for each write TDR.*
- volatile uint8\_t `bytesEachRead`  
*Bytes for each read RDR.*
- volatile uint8\_t `bytesLastRead`  
*Bytes for last read RDR.*
- volatile uint8\_t `isThereExtraRxBytes`  
*Is there extra RX byte.*
- uint8\_t \*volatile `txData`  
*Send buffer.*
- uint8\_t \*volatile `rxData`  
*Receive buffer.*
- volatile size\_t `txRemainingByteCount`  
*Number of bytes remaining to send.*
- volatile size\_t `rxRemainingByteCount`  
*Number of bytes remaining to receive.*
- volatile uint32\_t `writeRegRemainingTimes`  
*Write TDR register remaining times.*
- volatile uint32\_t `readRegRemainingTimes`  
*Read RDR register remaining times.*
- uint32\_t `totalByteCount`  
*Number of transfer bytes.*
- uint32\_t `txBuffIfNull`  
*Used if there is not txData for DMA purpose.*

- `uint32_t rxBuffIfNull`  
*Used if there is not rxData for DMA purpose.*
- `uint32_t transmitCommand`  
*Used to write TCR for DMA purpose.*
- `volatile uint8_t state`  
*LPSPI transfer state , `_lpspi_transfer_state`.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `lpspi_master_edma_transfer_callback_t callback`  
*Completion callback.*
- `void * userData`  
*Callback user data.*
- `edma_handle_t * edmaRxRegToRxDataHandle`  
*`edma_handle_t` handle point used for RxReg to RxData buff*
- `edma_handle_t * edmaTxDataToTxRegHandle`  
*`edma_handle_t` handle point used for TxData to TxReg buff*
- `edma_tcd_t lpspiSoftwareTCD [3]`  
*SoftwareTCD, internal used.*

## LPSPI eDMA Driver

### 30.3.2.1.0.22 Field Documentation

- 30.3.2.1.0.22.1 `volatile bool lpspi_master_edma_handle_t::isPcsContinuous`
  - 30.3.2.1.0.22.2 `volatile bool lpspi_master_edma_handle_t::isByteSwap`
  - 30.3.2.1.0.22.3 `volatile uint8_t lpspi_master_edma_handle_t::fifoSize`
  - 30.3.2.1.0.22.4 `volatile uint8_t lpspi_master_edma_handle_t::rxWatermark`
  - 30.3.2.1.0.22.5 `volatile uint8_t lpspi_master_edma_handle_t::bytesEachWrite`
  - 30.3.2.1.0.22.6 `volatile uint8_t lpspi_master_edma_handle_t::bytesEachRead`
  - 30.3.2.1.0.22.7 `volatile uint8_t lpspi_master_edma_handle_t::bytesLastRead`
  - 30.3.2.1.0.22.8 `volatile uint8_t lpspi_master_edma_handle_t::isThereExtraRxBytes`
  - 30.3.2.1.0.22.9 `uint8_t* volatile lpspi_master_edma_handle_t::txData`
  - 30.3.2.1.0.22.10 `uint8_t* volatile lpspi_master_edma_handle_t::rxData`
  - 30.3.2.1.0.22.11 `volatile size_t lpspi_master_edma_handle_t::txRemainingByteCount`
  - 30.3.2.1.0.22.12 `volatile size_t lpspi_master_edma_handle_t::rxRemainingByteCount`
  - 30.3.2.1.0.22.13 `volatile uint32_t lpspi_master_edma_handle_t::writeRegRemainingTimes`
  - 30.3.2.1.0.22.14 `volatile uint32_t lpspi_master_edma_handle_t::readRegRemainingTimes`
  - 30.3.2.1.0.22.15 `uint32_t lpspi_master_edma_handle_t::txBuffIfNull`
  - 30.3.2.1.0.22.16 `uint32_t lpspi_master_edma_handle_t::rxBuffIfNull`
  - 30.3.2.1.0.22.17 `uint32_t lpspi_master_edma_handle_t::transmitCommand`
  - 30.3.2.1.0.22.18 `volatile uint8_t lpspi_master_edma_handle_t::state`
  - 30.3.2.1.0.22.19 `uint8_t lpspi_master_edma_handle_t::nbytes`
  - 30.3.2.1.0.22.20 `lpspi_master_edma_transfer_callback_t lpspi_master_edma_handle_t::callback`
  - 30.3.2.1.0.22.21 `void* lpspi_master_edma_handle_t::userData`
- ### 30.3.2.2 struct `_lpspi_slave_edma_handle`

Forward declaration of the [\\_lpspi\\_slave\\_edma\\_handle](#) typedefs.



## Data Fields

- volatile bool `isByteSwap`  
*A flag that whether should byte swap.*
- volatile uint8\_t `fifoSize`  
*FIFO dataSize.*
- volatile uint8\_t `rxWatermark`  
*Rx watermark.*
- volatile uint8\_t `bytesEachWrite`  
*Bytes for each write TDR.*
- volatile uint8\_t `bytesEachRead`  
*Bytes for each read RDR.*
- volatile uint8\_t `bytesLastRead`  
*Bytes for last read RDR.*
- volatile uint8\_t `isThereExtraRxBytes`  
*Is there extra RX byte.*
- uint8\_t `nbytes`  
*eDMA minor byte transfer count initially configured.*
- uint8\_t \*volatile `txData`  
*Send buffer.*
- uint8\_t \*volatile `rxData`  
*Receive buffer.*
- volatile size\_t `txRemainingByteCount`  
*Number of bytes remaining to send.*
- volatile size\_t `rxRemainingByteCount`  
*Number of bytes remaining to receive.*
- volatile uint32\_t `writeRegRemainingTimes`  
*Write TDR register remaining times.*
- volatile uint32\_t `readRegRemainingTimes`  
*Read RDR register remaining times.*
- uint32\_t `totalByteCount`  
*Number of transfer bytes.*
- uint32\_t `txBuffIfNull`  
*Used if there is not txData for DMA purpose.*
- uint32\_t `rxBuffIfNull`  
*Used if there is not rxData for DMA purpose.*
- volatile uint8\_t `state`  
*LPSPI transfer state.*
- uint32\_t `errorCount`  
*Error count for slave transfer.*
- `lpspi_slave_edma_transfer_callback_t` callback  
*Completion callback.*
- void \* `userData`  
*Callback user data.*
- `edma_handle_t` \* `edmaRxRegToRxDataHandle`  
*edma\_handle\_t handle point used for RxReg to RxData buff*
- `edma_handle_t` \* `edmaTxDataToTxRegHandle`  
*edma\_handle\_t handle point used for TxData to TxReg*
- `edma_tcd_t` `lpspiSoftwareTCD` [2]  
*SoftwareTCD, internal used.*



**30.3.2.2.0.23 Field Documentation**

- 30.3.2.2.0.23.1 `volatile bool lpspi_slave_edma_handle_t::isByteSwap`
- 30.3.2.2.0.23.2 `volatile uint8_t lpspi_slave_edma_handle_t::fifoSize`
- 30.3.2.2.0.23.3 `volatile uint8_t lpspi_slave_edma_handle_t::rxWatermark`
- 30.3.2.2.0.23.4 `volatile uint8_t lpspi_slave_edma_handle_t::bytesEachWrite`
- 30.3.2.2.0.23.5 `volatile uint8_t lpspi_slave_edma_handle_t::bytesEachRead`
- 30.3.2.2.0.23.6 `volatile uint8_t lpspi_slave_edma_handle_t::bytesLastRead`
- 30.3.2.2.0.23.7 `volatile uint8_t lpspi_slave_edma_handle_t::isThereExtraRxBytes`
- 30.3.2.2.0.23.8 `uint8_t lpspi_slave_edma_handle_t::nbytes`
- 30.3.2.2.0.23.9 `uint8_t* volatile lpspi_slave_edma_handle_t::txData`
- 30.3.2.2.0.23.10 `uint8_t* volatile lpspi_slave_edma_handle_t::rxData`
- 30.3.2.2.0.23.11 `volatile size_t lpspi_slave_edma_handle_t::txRemainingByteCount`
- 30.3.2.2.0.23.12 `volatile size_t lpspi_slave_edma_handle_t::rxRemainingByteCount`
- 30.3.2.2.0.23.13 `volatile uint32_t lpspi_slave_edma_handle_t::writeRegRemainingTimes`
- 30.3.2.2.0.23.14 `volatile uint32_t lpspi_slave_edma_handle_t::readRegRemainingTimes`
- 30.3.2.2.0.23.15 `uint32_t lpspi_slave_edma_handle_t::txBuffIfNull`
- 30.3.2.2.0.23.16 `uint32_t lpspi_slave_edma_handle_t::rxBuffIfNull`
- 30.3.2.2.0.23.17 `volatile uint8_t lpspi_slave_edma_handle_t::state`
- 30.3.2.2.0.23.18 `uint32_t lpspi_slave_edma_handle_t::errorCount`
- 30.3.2.2.0.23.19 `lpspi_slave_edma_transfer_callback_t lpspi_slave_edma_handle_t::callback`
- 30.3.2.2.0.23.20 `void* lpspi_slave_edma_handle_t::userData`

**30.3.3 Macro Definition Documentation**

- 30.3.3.1 `#define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

**30.3.4 Typedef Documentation**

- 30.3.4.1 `typedef void(* lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)`

## LPSPI eDMA Driver

### Parameters

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

### 30.3.4.2 typedef void(\* lpspi\_slave\_edma\_transfer\_callback\_t)(LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, status\_t status, void \*userData)

### Parameters

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

## 30.3.5 Function Documentation

### 30.3.5.1 void LPSPI\_MasterTransferCreateHandleEDMA ( LPSPI\_Type \* base, lpspi\_master\_edma\_handle\_t \* handle, lpspi\_master\_edma\_transfer\_callback\_t callback, void \* userData, edma\_handle\_t \* edmaRxRegToRxDataHandle, edma\_handle\_t \* edmaTxDataToTxRegHandle )

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

Note that the LPSPI eDMA has a separated (Rx and Rx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaIntermediaryToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle.

### Parameters

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

|                                  |                                                                                 |
|----------------------------------|---------------------------------------------------------------------------------|
| <i>handle</i>                    | LPSPI handle pointer to <code>lpspi_master_edma_handle_t</code> .               |
| <i>callback</i>                  | LPSPI callback.                                                                 |
| <i>userData</i>                  | callback function parameter.                                                    |
| <i>edmaRxRegTo-RxDataHandle</i>  | <code>edmaRxRegToRxDataHandle</code> pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxData-ToTxReg-Handle</i> | <code>edmaTxDataToTxRegHandle</code> pointer to <a href="#">edma_handle_t</a> . |

### 30.3.5.2 `status_t LPSPI_MasterTransferEDMA ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle, lpspi_transfer_t * transfer )`

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

Note: The transfer data size should be an integer multiple of `bytesPerFrame` if `bytesPerFrame` is less than or equal to 4. For `bytesPerFrame` greater than 4: The transfer data size should be equal to `bytesPerFrame` if the `bytesPerFrame` is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of `bytesPerFrame`.

Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                                                |
| <i>handle</i>   | pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                                        |

Returns

status of `status_t`.

### 30.3.5.3 `void LPSPI_MasterTransferAbortEDMA ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle )`

This function aborts a transfer which is using eDMA.

Parameters

## LPSPI eDMA Driver

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

### 30.3.5.4 `status_t LPSPI_MasterTransferGetCountEDMA ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle, size_t * count )`

This function gets the master eDMA transfer remaining bytes.

Parameters

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

Returns

status of `status_t`.

### 30.3.5.5 `void LPSPI_SlaveTransferCreateHandleEDMA ( LPSPI_Type * base, lpspi_slave_edma_handle_t * handle, lpspi_slave_edma_transfer_callback_t callback, void * userData, edma_handle_t * edmaRxRegToRxDataHandle, edma_handle_t * edmaTxDataToTxRegHandle )`

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

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for `edmaRxRegToRxDataHandle` and Tx DMAMUX source for `edmaTxDataToTxRegHandle`. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for `edmaRxRegToRxDataHandle` .

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                   |
| <i>handle</i> | LPSPI handle pointer to <code>lpspi_slave_edma_handle_t</code> . |

|                                  |                                                                    |
|----------------------------------|--------------------------------------------------------------------|
| <i>callback</i>                  | LPSPI callback.                                                    |
| <i>userData</i>                  | callback function parameter.                                       |
| <i>edmaRxRegTo-RxDataHandle</i>  | edmaRxRegToRxDataHandle pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxData-ToTxReg-Handle</i> | edmaTxDataToTxRegHandle pointer to <a href="#">edma_handle_t</a> . |

### 30.3.5.6 **status\_t LPSPI\_SlaveTransferEDMA ( LPSPI\_Type \* *base*, lpspi\_slave\_edma\_handle\_t \* *handle*, lpspi\_transfer\_t \* *transfer* )**

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                                                  |
| <i>handle</i>   | pointer to <a href="#">lpspi_slave_edma_handle_t</a> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                                          |

Returns

status of [status\\_t](#).

### 30.3.5.7 **void LPSPI\_SlaveTransferAbortEDMA ( LPSPI\_Type \* *base*, lpspi\_slave\_edma\_handle\_t \* *handle* )**

This function aborts a transfer which is using eDMA.

Parameters

## LPSPI eDMA Driver

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

### 30.3.5.8 `status_t LPSPI_SlaveTransferGetCountEDMA ( LPSPI_Type * base, lpspi_slave_edma_handle_t * handle, size_t * count )`

This function gets the slave eDMA transfer remaining bytes.

Parameters

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

Returns

status of `status_t`.



## 30.4 LPSPI FreeRTOS Driver

### 30.4.1 Overview

#### Driver version

- #define `FSL_LPSPI_FREERTOS_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*LPSPI freertos driver version 2.0.2.*

#### LPSPI RTOS Operation

- status\_t `LPSPI_RTOS_Init` (lpspi\_rtos\_handle\_t \*handle, LPSPI\_Type \*base, const lpspi\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes LPSPI.*
- status\_t `LPSPI_RTOS_Deinit` (lpspi\_rtos\_handle\_t \*handle)  
*Deinitializes the LPSPI.*
- status\_t `LPSPI_RTOS_Transfer` (lpspi\_rtos\_handle\_t \*handle, lpspi\_transfer\_t \*transfer)  
*Performs SPI transfer.*

### 30.4.2 Macro Definition Documentation

#### 30.4.2.1 #define FSL\_LPSPI\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

### 30.4.3 Function Documentation

#### 30.4.3.1 status\_t LPSPI\_RTOS\_Init ( lpspi\_rtos\_handle\_t \* handle, LPSPI\_Type \* base, const lpspi\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )

This function initializes the LPSPI module and related RTOS context.

Parameters

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

Returns

status of the operation.

### 30.4.3.2 `status_t LPSPI_RTOS_Deinit ( lpspi_rtos_handle_t * handle )`

This function deinitializes the LPSPI module and related RTOS context.

## Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPSPI handle. |
|---------------|------------------------|

### 30.4.3.3 `status_t LPSPI_RTOS_Transfer ( lpspi_rtos_handle_t * handle, lpspi_transfer_t * transfer )`

This function performs an SPI transfer according to data given in the transfer structure.

## Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS LPSPI handle.                        |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.





## Chapter 31

# LPUART: Low Power UART Driver

### 31.1 Overview

#### Modules

- [LPUART DMA Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)
- [LPUART eDMA Driver](#)

## LPUART Driver

### 31.2 LPUART Driver

#### 31.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

#### 31.2.2 Typical use case

##### 31.2.2.1 LPUART Operation

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

### Data Structures

- struct [lpuart\\_config\\_t](#)  
*LPUART configuration structure. [More...](#)*
- struct [lpuart\\_transfer\\_t](#)  
*LPUART transfer structure. [More...](#)*
- struct [lpuart\\_handle\\_t](#)  
*LPUART handle structure. [More...](#)*

### 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` = MAKE\_STATUS(kStatusGroup\_LPUART, 6),
  - `kStatus_LPUART_Error` = MAKE\_STATUS(kStatusGroup\_LPUART, 7),
  - `kStatus_LPUART_RxRingBufferOverrun`,
  - `kStatus_LPUART_RxHardwareOverrun` = MAKE\_STATUS(kStatusGroup\_LPUART, 9),
  - `kStatus_LPUART_NoiseError` = MAKE\_STATUS(kStatusGroup\_LPUART, 10),
  - `kStatus_LPUART_FramingError` = MAKE\_STATUS(kStatusGroup\_LPUART, 11),
  - `kStatus_LPUART_ParityError` = MAKE\_STATUS(kStatusGroup\_LPUART, 12),
  - `kStatus_LPUART_BaudrateNotSupport`,
  - `kStatus_LPUART_IdleLineDetected` = MAKE\_STATUS(kStatusGroup\_LPUART, 14) }

*Error codes for the LPUART driver.*
- enum `lpuart_parity_mode_t` {
  - `kLPUART_ParityDisabled` = 0x0U,
  - `kLPUART_ParityEven` = 0x2U,
  - `kLPUART_ParityOdd` = 0x3U }

*LPUART parity mode.*
- enum `lpuart_data_bits_t` {
  - `kLPUART_EightDataBits` = 0x0U,
  - `kLPUART_SevenDataBits` = 0x1U }

*LPUART data bits count.*
- enum `lpuart_stop_bit_count_t` {
  - `kLPUART_OneStopBit` = 0U,
  - `kLPUART_TwoStopBit` = 1U }

*LPUART stop bit count.*
- enum `lpuart_transmit_cts_source_t` {
  - `kLPUART_CtsSourcePin` = 0U,
  - `kLPUART_CtsSourceMatchResult` = 1U }

*LPUART transmit CTS source.*
- enum `lpuart_transmit_cts_config_t` {
  - `kLPUART_CtsSampleAtStart` = 0U,
  - `kLPUART_CtsSampleAtIdle` = 1U }

*LPUART transmit CTS configure.*
- enum `lpuart_idle_type_select_t` {
  - `kLPUART_IdleTypeStartBit` = 0U,
  - `kLPUART_IdleTypeStopBit` = 1U }

*LPUART idle flag type defines when the receiver starts counting.*
- enum `lpuart_idle_config_t` {

## LPUART Driver

```
kLPUART_IdleCharacter1 = 0U,  
kLPUART_IdleCharacter2 = 1U,  
kLPUART_IdleCharacter4 = 2U,  
kLPUART_IdleCharacter8 = 3U,  
kLPUART_IdleCharacter16 = 4U,  
kLPUART_IdleCharacter32 = 5U,  
kLPUART_IdleCharacter64 = 6U,  
kLPUART_IdleCharacter128 = 7U }
```

*LPUART idle detected configuration.*

- enum `_lpuart_interrupt_enable` {  
    kLPUART\_LinBreakInterruptEnable = (LPUART\_BAUD\_LBKDIE\_MASK >> 8),  
    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),  
    kLPUART\_TxFifoOverflowInterruptEnable = (LPUART\_FIFO\_TXOFE\_MASK >> 8),  
    kLPUART\_RxFifoUnderflowInterruptEnable = (LPUART\_FIFO\_RXUFE\_MASK >> 8) }

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

- enum `_lpuart_flags` {  
    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\_LinBreakFlag = (LPUART\_STAT\_LBKDIF\_MASK),  
    kLPUART\_RxActiveEdgeFlag,  
    kLPUART\_RxActiveFlag,  
    kLPUART\_DataMatch1Flag = LPUART\_STAT\_MA1F\_MASK,  
    kLPUART\_DataMatch2Flag = LPUART\_STAT\_MA2F\_MASK,  
    kLPUART\_NoiseErrorInRxDataRegFlag,  
    kLPUART\_ParityErrorInRxDataRegFlag,  
    kLPUART\_TxFifoEmptyFlag = (LPUART\_FIFO\_TXEMPT\_MASK >> 16),  
    kLPUART\_RxFifoEmptyFlag = (LPUART\_FIFO\_RXEMPT\_MASK >> 16),  
    kLPUART\_TxFifoOverflowFlag,  
    kLPUART\_RxFifoUnderflowFlag }

*LPUART status flags.*



## Driver version

- #define [FSL\\_LPUART\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 5))  
*LPUART driver version 2.2.5.*

## Software Reset

- static void [LPUART\\_SoftwareReset](#) (LPUART\_Type \*base)  
*Resets the LPUART using software.*

## Initialization and deinitialization

- status\_t [LPUART\\_Init](#) (LPUART\_Type \*base, const [lpuart\\_config\\_t](#) \*config, uint32\_t srcClock\_Hz)  
*Initializes an LPUART instance with the user configuration structure and the peripheral clock.*
- void [LPUART\\_Deinit](#) (LPUART\_Type \*base)  
*Deinitializes a LPUART instance.*
- void [LPUART\\_GetDefaultConfig](#) ([lpuart\\_config\\_t](#) \*config)  
*Gets the default configuration structure.*
- 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.*
- static uint32\_t [LPUART\\_GetDataRegisterAddress](#) (LPUART\_Type \*base)  
*Gets the LPUART data register address.*
- static void [LPUART\\_EnableTxDMA](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART transmitter DMA request.*
- static void [LPUART\\_EnableRxDMA](#) (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver DMA.*

### Bus Operations

- `uint32_t LPUART_GetInstance` (LPUART\_Type \*base)  
*Get the LPUART instance from peripheral base address.*
- `static void LPUART_EnableTx` (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART transmitter.*
- `static void LPUART_EnableRx` (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver.*
- `static void LPUART_WriteByte` (LPUART\_Type \*base, uint8\_t data)  
*Writes to the transmitter register.*
- `static uint8_t LPUART_ReadByte` (LPUART\_Type \*base)  
*Reads the receiver register.*
- `void LPUART_WriteBlocking` (LPUART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the transmitter register using a blocking method.*
- `status_t LPUART_ReadBlocking` (LPUART\_Type \*base, uint8\_t \*data, size\_t length)  
*Reads the receiver data register using a blocking method.*

### Transactional

- `void LPUART_TransferCreateHandle` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, lpuart\_transfer\_callback\_t callback, void \*userData)  
*Initializes the LPUART handle.*
- `status_t LPUART_TransferSendNonBlocking` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, lpuart\_transfer\_t \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- `void LPUART_TransferStartRingBuffer` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- `void LPUART_TransferStopRingBuffer` (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- `size_t LPUART_TransferGetRxRingBufferLength` (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Get the length of received data in RX ring buffer.*
- `void LPUART_TransferAbortSend` (LPUART\_Type \*base, lpuart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- `status_t LPUART_TransferGetSendCount` (LPUART\_Type \*base, lpuart\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes that have been written to the LPUART transmitter 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)  
*Gets 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.*

### 31.2.3 Data Structure Documentation

#### 31.2.3.1 struct lpuart\_config\_t

##### Data Fields

- uint32\_t `baudRate_Bps`  
*LPUART baud rate.*
- `lpuart_parity_mode_t` `parityMode`  
*Parity mode, disabled (default), even, odd.*
- `lpuart_data_bits_t` `dataBitsCount`  
*Data bits count, eight (default), seven.*
- bool `isMsb`  
*Data bits order, LSB (default), MSB.*
- `lpuart_stop_bit_count_t` `stopBitCount`  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- uint8\_t `txFifoWatermark`  
*TX FIFO watermark.*
- uint8\_t `rxFifoWatermark`  
*RX FIFO watermark.*
- bool `enableRxRTS`  
*RX RTS enable.*
- bool `enableTxCTS`  
*TX CTS enable.*
- `lpuart_transmit_cts_source_t` `txCtsSource`  
*TX CTS source.*
- `lpuart_transmit_cts_config_t` `txCtsConfig`  
*TX CTS configure.*
- `lpuart_idle_type_select_t` `rxIdleType`  
*RX IDLE type.*
- `lpuart_idle_config_t` `rxIdleConfig`  
*RX IDLE configuration.*
- bool `enableTx`  
*Enable TX.*
- bool `enableRx`  
*Enable RX.*

## LPUART Driver

### 31.2.3.1.0.24 Field Documentation

#### 31.2.3.1.0.24.1 `lpuart_idle_type_select_t lpuart_config_t::rxIdleType`

#### 31.2.3.1.0.24.2 `lpuart_idle_config_t lpuart_config_t::rxIdleConfig`

### 31.2.3.2 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.*

### 31.2.3.2.0.25 Field Documentation

#### 31.2.3.2.0.25.1 `uint8_t* lpuart_transfer_t::data`

#### 31.2.3.2.0.25.2 `size_t lpuart_transfer_t::dataSize`

### 31.2.3.3 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`  
*Size of the data to receive.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*
- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `lpuart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*LPUART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*

- volatile uint8\_t **rxState**  
*RX transfer state.*
- bool **isSevenDataBits**  
*Seven data bits flag.*

## LPUART Driver

### 31.2.3.3.0.26 Field Documentation

- 31.2.3.3.0.26.1 `uint8_t* volatile lpuart_handle_t::txData`
- 31.2.3.3.0.26.2 `volatile size_t lpuart_handle_t::txDataSize`
- 31.2.3.3.0.26.3 `size_t lpuart_handle_t::txDataSizeAll`
- 31.2.3.3.0.26.4 `uint8_t* volatile lpuart_handle_t::rxData`
- 31.2.3.3.0.26.5 `volatile size_t lpuart_handle_t::rxDataSize`
- 31.2.3.3.0.26.6 `size_t lpuart_handle_t::rxDataSizeAll`
- 31.2.3.3.0.26.7 `uint8_t* lpuart_handle_t::rxRingBuffer`
- 31.2.3.3.0.26.8 `size_t lpuart_handle_t::rxRingBufferSize`
- 31.2.3.3.0.26.9 `volatile uint16_t lpuart_handle_t::rxRingBufferHead`
- 31.2.3.3.0.26.10 `volatile uint16_t lpuart_handle_t::rxRingBufferTail`
- 31.2.3.3.0.26.11 `lpuart_transfer_callback_t lpuart_handle_t::callback`
- 31.2.3.3.0.26.12 `void* lpuart_handle_t::userData`
- 31.2.3.3.0.26.13 `volatile uint8_t lpuart_handle_t::txState`
- 31.2.3.3.0.26.14 `volatile uint8_t lpuart_handle_t::rxState`
- 31.2.3.3.0.26.15 `bool lpuart_handle_t::isSevenDataBits`

### 31.2.4 Macro Definition Documentation

- 31.2.4.1 `#define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 2, 5))`

### 31.2.5 Typedef Documentation

- 31.2.5.1 `typedef void(* lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle, status_t status, void *userData)`

### 31.2.6 Enumeration Type Documentation

- 31.2.6.1 `enum _lpuart_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.  
*kStatus\_LPUART\_IdleLineDetected* IDLE flag.

### 31.2.6.2 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.

### 31.2.6.3 enum lpuart\_data\_bits\_t

Enumerator

*kLPUART\_EightDataBits* Eight data bit.  
*kLPUART\_SevenDataBits* Seven data bit.

### 31.2.6.4 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLPUART\_OneStopBit* One stop bit.  
*kLPUART\_TwoStopBit* Two stop bits.

### 31.2.6.5 enum lpuart\_transmit\_cts\_source\_t

Enumerator

*kLPUART\_CtsSourcePin* CTS resource is the LPUART\_CTS pin.  
*kLPUART\_CtsSourceMatchResult* CTS resource is the match result.

## LPUART Driver

### 31.2.6.6 enum lpuart\_transmit\_cts\_config\_t

Enumerator

*kLPUART\_CtsSampleAtStart* CTS input is sampled at the start of each character.

*kLPUART\_CtsSampleAtIdle* CTS input is sampled when the transmitter is idle.

### 31.2.6.7 enum lpuart\_idle\_type\_select\_t

Enumerator

*kLPUART\_IdleTypeStartBit* Start counting after a valid start bit.

*kLPUART\_IdleTypeStopBit* Start counting after a stop bit.

### 31.2.6.8 enum lpuart\_idle\_config\_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

*kLPUART\_IdleCharacter1* the number of idle characters.

*kLPUART\_IdleCharacter2* the number of idle characters.

*kLPUART\_IdleCharacter4* the number of idle characters.

*kLPUART\_IdleCharacter8* the number of idle characters.

*kLPUART\_IdleCharacter16* the number of idle characters.

*kLPUART\_IdleCharacter32* the number of idle characters.

*kLPUART\_IdleCharacter64* the number of idle characters.

*kLPUART\_IdleCharacter128* the number of idle characters.

### 31.2.6.9 enum \_lpuart\_interrupt\_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

*kLPUART\_LinBreakInterruptEnable* LIN break detect.

*kLPUART\_RxActiveEdgeInterruptEnable* Receive Active Edge.

*kLPUART\_TxDataRegEmptyInterruptEnable* Transmit data register empty.

*kLPUART\_TransmissionCompleteInterruptEnable* Transmission complete.

*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.  
***kLPUART\_TxFifoOverflowInterruptEnable*** Transmit FIFO Overflow.  
***kLPUART\_RxFifoUnderflowInterruptEnable*** Receive FIFO Underflow.

### 31.2.6.10 enum \_lpuart\_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

***kLPUART\_TxDataRegEmptyFlag*** Transmit data register empty flag, sets when transmit buffer is empty.  
***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\_LinBreakFlag*** LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.  
***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.  
***kLPUART\_DataMatch1Flag*** The next character to be read from LPUART\_DATA matches MA1.  
***kLPUART\_DataMatch2Flag*** The next character to be read from LPUART\_DATA matches MA2.  
***kLPUART\_NoiseErrorInRxDataRegFlag*** NOISY bit, sets if noise detected in current data word.  
***kLPUART\_ParityErrorInRxDataRegFlag*** PARITYE bit, sets if noise detected in current data word.  
***kLPUART\_TxFifoEmptyFlag*** TXEMPT bit, sets if transmit buffer is empty.  
***kLPUART\_RxFifoEmptyFlag*** RXEMPT bit, sets if receive buffer is empty.  
***kLPUART\_TxFifoOverflowFlag*** TXOF bit, sets if transmit buffer overflow occurred.  
***kLPUART\_RxFifoUnderflowFlag*** RXUF bit, sets if receive buffer underflow occurred.

### 31.2.7 Function Documentation

**31.2.7.1** `static void LPUART_SoftwareReset ( LPUART_Type * base ) [inline],  
[static]`

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

## Parameters

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

### 31.2.7.2 `status_t LPUART_Init ( LPUART_Type * base, const lpuart_config_t * config, uint32_t srcClock_Hz )`

This function configures the LPUART module with user-defined settings. Call the [LPUART\\_GetDefault-Config\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

## Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>base</i>        | LPUART peripheral base address.                    |
| <i>config</i>      | Pointer to a user-defined configuration structure. |
| <i>srcClock_Hz</i> | LPUART clock source frequency in HZ.               |

## Return values

|                                           |                                                  |
|-------------------------------------------|--------------------------------------------------|
| <i>kStatus_LPUART_-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                    | LPUART initialize succeed                        |

### 31.2.7.3 `void LPUART_Deinit ( LPUART_Type * base )`

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

## Parameters

## LPUART Driver

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

### 31.2.7.4 void LPUART\_GetDefaultConfig ( lpuart\_config\_t \* config )

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate\_Bps = 115200U; lpuartConfig->parityMode = kLPUART\_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART\_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART\_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART\_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART\_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

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

### 31.2.7.5 status\_t LPUART\_SetBaudRate ( LPUART\_Type \* base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz )

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 200000000U);  
*
```

Parameters

|                     |                                      |
|---------------------|--------------------------------------|
| <i>base</i>         | LPUART peripheral base address.      |
| <i>baudRate_Bps</i> | LPUART baudrate to be set.           |
| <i>srcClock_Hz</i>  | LPUART clock source frequency in HZ. |

Return values

|                                          |                                                        |
|------------------------------------------|--------------------------------------------------------|
| <i>kStatus_LPUART_BaudrateNotSupport</i> | Baudrate is not supported in the current clock source. |
| <i>kStatus_Success</i>                   | Set baudrate succeeded.                                |

### 31.2.7.6 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`.

### 31.2.7.7 `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 effect.

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

## LPUART Driver

|                        |                                 |
|------------------------|---------------------------------|
| <i>kStatus_Success</i> | Status in the mask are cleared. |
|------------------------|---------------------------------|

### 31.2.7.8 void LPUART\_EnableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the [\\_lpuart\\_interrupt\\_enable](#). This examples shows how to enable TX empty interrupt and RX full interrupt:

```
* LPUART_EnableInterrupts(LPUART1,  
kLPUART_TxDataRegEmptyInterruptEnable |  
kLPUART_RxDataRegFullInterruptEnable);  
*
```

#### Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                  |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_uart_interrupt_enable</a> . |

### 31.2.7.9 void LPUART\_DisableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_lpuart\\_interrupt\\_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
* LPUART_DisableInterrupts(LPUART1,  
kLPUART_TxDataRegEmptyInterruptEnable |  
kLPUART_RxDataRegFullInterruptEnable);  
*
```

#### Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> . |

### 31.2.7.10 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](#).

### 31.2.7.11 static uint32\_t LPUART\_GetDataRegisterAddress ( LPUART\_Type \* *base* ) [inline], [static]

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

## Parameters

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

## Returns

LPUART data register addresses which are used both by the transmitter and receiver.

### 31.2.7.12 static void LPUART\_EnableTxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

### 31.2.7.13 static void LPUART\_EnableRxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

## LPUART Driver

### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

### 31.2.7.14 `uint32_t LPUART_GetInstance ( LPUART_Type * base )`

### Parameters

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

### Returns

LPUART instance.

### 31.2.7.15 `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. |

### 31.2.7.16 `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. |

### 31.2.7.17 `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.  |

### 31.2.7.18 `static uint8_t LPUART_ReadByte ( LPUART_Type * base ) [inline], [static]`

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

## Parameters

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

## Returns

Data read from data register.

### 31.2.7.19 `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 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.

## 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.          |

### 31.2.7.20 `status_t LPUART_ReadBlocking ( LPUART_Type * base, uint8_t * data, size_t length )`

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

## LPUART Driver

### 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.                 |

### 31.2.7.21 void LPUART\_TransferCreateHandle ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, lpuart\_transfer\_callback\_t *callback*, void \* *userData* )

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

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as *ringBuffer*.

### Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | LPUART peripheral base address. |
| <i>handle</i>   | LPUART handle pointer.          |
| <i>callback</i> | Callback function.              |
| <i>userData</i> | User data.                      |

### 31.2.7.22 `status_t LPUART_TransferSendNonBlocking ( LPUART_Type * base, lpuart_handle_t * handle, lpuart_transfer_t * xfer )`

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the `kStatus_LPUART_TxIdle` as status parameter.

#### Note

The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

#### Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                    |
| <i>handle</i> | LPUART handle pointer.                                             |
| <i>xfer</i>   | LPUART transfer structure, see <a href="#">lpuart_transfer_t</a> . |

#### Return values

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

### 31.2.7.23 `void LPUART_TransferStartRingBuffer ( LPUART_Type * base, lpuart_handle_t * handle, uint8_t * ringBuffer, size_t ringBufferSize )`

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

#### Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

## LPUART Driver

### 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.                                                                     |

### 31.2.7.24 void LPUART\_TransferStopRingBuffer ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

### Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 31.2.7.25 size\_t LPUART\_TransferGetRxRingBufferLength ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )

handle LPUART handle pointer.

### Returns

Length of received data in RX ring buffer.

### 31.2.7.26 void LPUART\_TransferAbortSend ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )

This function aborts the interrupt driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

### Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 31.2.7.27 **status\_t** LPUART\_TransferGetSendCount ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been written to LPUART TX register by an interrupt method.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

### 31.2.7.28 **status\_t** LPUART\_TransferReceiveNonBlocking ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter *kStatus\_UART\_RxIdle*. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to *xfer->data*, which returns with the parameter *receivedBytes* set to 5. For the remaining 5 bytes, the newly arrived data is saved from *xfer->data[5]*. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to *xfer->data*. When all data is received, the upper layer is notified.

## LPUART Driver

### 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

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

### 31.2.7.29 void LPUART\_TransferAbortReceive ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle* )

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

### Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 31.2.7.30 status\_t LPUART\_TransferGetReceiveCount ( LPUART\_Type \* *base*, lpuart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

### Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

**31.2.7.31 void LPUART\_TransferHandleIRQ ( LPUART\_Type \* *base*, lpuart\_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.          |

**31.2.7.32 void LPUART\_TransferHandleErrorIRQ ( LPUART\_Type \* *base*, lpuart\_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.          |

## LPUART DMA Driver

### 31.3 LPUART DMA Driver

#### 31.3.1 Overview

#### Data Structures

- struct [lpuart\\_dma\\_handle\\_t](#)  
*LPUART DMA handle. [More...](#)*

#### Typedefs

- typedef void(\* [lpuart\\_dma\\_transfer\\_callback\\_t](#) )(LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, status\_t status, void \*userData)  
*LPUART transfer callback function.*

#### Driver version

- #define [FSL\\_LPUART\\_DMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 5))  
*LPUART DMA driver version 2.2.5.*

#### EDMA transactional

- void [LPUART\\_TransferCreateHandleDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, [lpuart\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, dma\_handle\_t \*txDmaHandle, dma\_handle\_t \*rxDmaHandle)  
*Initializes the LPUART handle which is used in transactional functions.*
- status\_t [LPUART\\_TransferSendDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Sends data using DMA.*
- status\_t [LPUART\\_TransferReceiveDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Receives data using DMA.*
- void [LPUART\\_TransferAbortSendDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle)  
*Aborts the sent data using DMA.*
- void [LPUART\\_TransferAbortReceiveDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle)  
*Aborts the received data using DMA.*
- status\_t [LPUART\\_TransferGetSendCountDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the LPUART TX register.*
- status\_t [LPUART\\_TransferGetReceiveCountDMA](#) (LPUART\_Type \*base, lpuart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of received bytes.*



## 31.3.2 Data Structure Documentation

### 31.3.2.1 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.*

## LPUART DMA Driver

### 31.3.2.1.0.27 Field Documentation

31.3.2.1.0.27.1 `lpuart_dma_transfer_callback_t lpuart_dma_handle_t::callback`

31.3.2.1.0.27.2 `void* lpuart_dma_handle_t::userData`

31.3.2.1.0.27.3 `size_t lpuart_dma_handle_t::rxDataSizeAll`

31.3.2.1.0.27.4 `size_t lpuart_dma_handle_t::txDataSizeAll`

31.3.2.1.0.27.5 `dma_handle_t* lpuart_dma_handle_t::txDmaHandle`

31.3.2.1.0.27.6 `dma_handle_t* lpuart_dma_handle_t::rxDmaHandle`

31.3.2.1.0.27.7 `volatile uint8_t lpuart_dma_handle_t::txState`

### 31.3.3 Macro Definition Documentation

31.3.3.1 `#define FSL_LPUART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 5))`

### 31.3.4 Typedef Documentation

31.3.4.1 `typedef void(* lpuart_dma_transfer_callback_t)(LPUART_Type *base, lpuart_dma_handle_t *handle, status_t status, void *userData)`

### 31.3.5 Function Documentation

31.3.5.1 `void LPUART_TransferCreateHandleDMA ( LPUART_Type * base, lpuart_dma_handle_t * handle, lpuart_dma_transfer_callback_t callback, void * userData, dma_handle_t * txDmaHandle, dma_handle_t * rxDmaHandle )`

## Parameters

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <i>base</i>        | LPUART peripheral base address.                        |
| <i>handle</i>      | Pointer to <code>lpuart_dma_handle_t</code> 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.         |

### 31.3.5.2 `status_t LPUART_TransferSendDMA ( LPUART_Type * base, lpuart_dma_handle_t * handle, lpuart_transfer_t * xfer )`

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                        |
| <i>handle</i> | LPUART handle pointer.                                                 |
| <i>xfer</i>   | LPUART DMA transfer structure. See <a href="#">lpuart_transfer_t</a> . |

## Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 31.3.5.3 `status_t LPUART_TransferReceiveDMA ( LPUART_Type * base, lpuart_dma_handle_t * handle, lpuart_transfer_t * xfer )`

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

## Parameters

---

## LPUART DMA Driver

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                        |
| <i>handle</i> | Pointer to <code>lpuart_dma_handle_t</code> 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.           |

### 31.3.5.4 void LPUART\_TransferAbortSendDMA ( LPUART\_Type \* *base*, lpuart\_dma\_handle\_t \* *handle* )

This function aborts send data using DMA.

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address                        |
| <i>handle</i> | Pointer to <code>lpuart_dma_handle_t</code> structure |

### 31.3.5.5 void LPUART\_TransferAbortReceiveDMA ( LPUART\_Type \* *base*, lpuart\_dma\_handle\_t \* *handle* )

This function aborts the received data using DMA.

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address                        |
| <i>handle</i> | Pointer to <code>lpuart_dma_handle_t</code> structure |

### 31.3.5.6 status\_t LPUART\_TransferGetSendCountDMA ( LPUART\_Type \* *base*, lpuart\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been written to LPUART TX register by DMA.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

## Return values

|                                     |                                                             |
|-------------------------------------|-------------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                        |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

### 31.3.5.7 `status_t LPUART_TransferGetReceiveCountDMA ( LPUART_Type * base, lpuart_dma_handle_t * handle, uint32_t * count )`

This function gets the number of received bytes.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

## Return values

|                                     |                                                             |
|-------------------------------------|-------------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                                     |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

### 31.4 LPUART eDMA Driver

#### 31.4.1 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.*

#### Driver version

- #define `FSL_LPUART_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 5))  
*LPUART EDMA driver version 2.2.5.*

#### eDMA transactional

- void `LPUART_TransferCreateHandleEDMA` (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, lpuart\_edma\_transfer\_callback\_t callback, void \*userData, edma\_handle\_t \*txEdmaHandle, edma\_handle\_t \*rxEdmaHandle)  
*Initializes the LPUART handle which is used in transactional functions.*
- status\_t `LPUART_SendEDMA` (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, lpuart\_transfer\_t \*xfer)  
*Sends data using eDMA.*
- status\_t `LPUART_ReceiveEDMA` (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, lpuart\_transfer\_t \*xfer)  
*Receives data using eDMA.*
- void `LPUART_TransferAbortSendEDMA` (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void `LPUART_TransferAbortReceiveEDMA` (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the received data using eDMA.*
- status\_t `LPUART_TransferGetSendCountEDMA` (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the LPUART TX register.*
- status\_t `LPUART_TransferGetReceiveCountEDMA` (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of received bytes.*

## 31.4.2 Data Structure Documentation

### 31.4.2.1 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.*
- `uint8_t` `nbytes`  
*eDMA minor byte transfer count initially configured.*
- `volatile uint8_t` `txState`  
*TX transfer state.*
- `volatile uint8_t` `rxState`  
*RX transfer state.*

## LPUART eDMA Driver

### 31.4.2.1.0.28 Field Documentation

31.4.2.1.0.28.1 `lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback`

31.4.2.1.0.28.2 `void* lpuart_edma_handle_t::userData`

31.4.2.1.0.28.3 `size_t lpuart_edma_handle_t::rxDataSizeAll`

31.4.2.1.0.28.4 `size_t lpuart_edma_handle_t::txDataSizeAll`

31.4.2.1.0.28.5 `edma_handle_t* lpuart_edma_handle_t::txEdmaHandle`

31.4.2.1.0.28.6 `edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle`

31.4.2.1.0.28.7 `uint8_t lpuart_edma_handle_t::nbytes`

31.4.2.1.0.28.8 `volatile uint8_t lpuart_edma_handle_t::txState`

### 31.4.3 Macro Definition Documentation

31.4.3.1 `#define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 5))`

### 31.4.4 Typedef Documentation

31.4.4.1 `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData)`

### 31.4.5 Function Documentation

31.4.5.1 `void LPUART_TransferCreateHandleEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`



## Parameters

|                     |                                                         |
|---------------------|---------------------------------------------------------|
| <i>base</i>         | LPUART peripheral base address.                         |
| <i>handle</i>       | Pointer to <code>lpuart_edma_handle_t</code> structure. |
| <i>callback</i>     | Callback function.                                      |
| <i>userData</i>     | User data.                                              |
| <i>txEdmaHandle</i> | User requested DMA handle for TX DMA transfer.          |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.          |

### 31.4.5.2 `status_t LPUART_SendEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_transfer_t * xfer )`

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

## Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                         |
| <i>handle</i> | LPUART handle pointer.                                                  |
| <i>xfer</i>   | LPUART eDMA transfer structure. See <a href="#">lpuart_transfer_t</a> . |

## Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 31.4.5.3 `status_t LPUART_ReceiveEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, lpuart_transfer_t * xfer )`

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

## Parameters

---

## LPUART eDMA Driver

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                         |
| <i>handle</i> | Pointer to <code>lpuart_edma_handle_t</code> structure.                 |
| <i>xfer</i>   | LPUART eDMA transfer structure, see <a href="#">lpuart_transfer_t</a> . |

Return values

|                                |                            |
|--------------------------------|----------------------------|
| <i>kStatus_Success</i>         | if succeed, others fail.   |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous transfer ongoing. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.          |

### 31.4.5.4 void LPUART\_TransferAbortSendEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                         |
| <i>handle</i> | Pointer to <code>lpuart_edma_handle_t</code> structure. |

### 31.4.5.5 void LPUART\_TransferAbortReceiveEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the received data using eDMA.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                         |
| <i>handle</i> | Pointer to <code>lpuart_edma_handle_t</code> structure. |

### 31.4.5.6 status\_t LPUART\_TransferGetSendCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes written to the LPUART TX register by DMA.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

## Return values

|                                     |                                                             |
|-------------------------------------|-------------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                        |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

### 31.4.5.7 `status_t LPUART_TransferGetReceiveCountEDMA ( LPUART_Type * base, lpuart_edma_handle_t * handle, uint32_t * count )`

This function gets the number of received bytes.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Receive bytes count.            |

## Return values

|                                     |                                                             |
|-------------------------------------|-------------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                                     |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

## LPUART FreeRTOS Driver

### 31.5 LPUART FreeRTOS Driver

#### 31.5.1 Overview

##### Data Structures

- struct [lpuart\\_rtos\\_config\\_t](#)  
*LPUART RTOS configuration structure. [More...](#)*

##### Driver version

- #define [FSL\\_LPUART\\_FREERTOS\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 5))  
*LPUART freertos driver version 2.2.5.*

##### LPUART RTOS Operation

- int [LPUART\\_RTOS\\_Init](#) (lpuart\_rtos\_handle\_t \*handle, lpuart\_handle\_t \*t\_handle, const [lpuart\\_rtos\\_config\\_t](#) \*cfg)  
*Initializes an LPUART instance for operation in RTOS.*
- int [LPUART\\_RTOS\\_Deinit](#) (lpuart\_rtos\_handle\_t \*handle)  
*Deinitializes an LPUART instance for operation.*

##### LPUART transactional Operation

- int [LPUART\\_RTOS\\_Send](#) (lpuart\_rtos\_handle\_t \*handle, 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.*

#### 31.5.2 Data Structure Documentation

##### 31.5.2.1 struct lpuart\_rtos\_config\_t

###### Data Fields

- LPUART\_Type \* [base](#)  
*UART base address.*
- uint32\_t [srcclk](#)  
*UART source clock in Hz.*
- uint32\_t [baudrate](#)  
*Desired communication speed.*
- [lpuart\\_parity\\_mode\\_t](#) [parity](#)  
*Parity setting.*

- `lpuart_stop_bit_count_t` stopbits  
*Number of stop bits to use.*
- `uint8_t * buffer`  
*Buffer for background reception.*
- `uint32_t buffer_size`  
*Size of buffer for background reception.*

### 31.5.3 Macro Definition Documentation

31.5.3.1 `#define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 5))`

### 31.5.4 Function Documentation

31.5.4.1 `int LPUART_RTOS_Init ( lpuart_rtos_handle_t * handle, lpuart_handle_t * t_handle, const lpuart_rtos_config_t * cfg )`

Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS LPUART handle, the pointer to an allocated space for RTOS context.          |
| <i>t_handle</i> | The pointer to an allocated space to store the transactional layer internal state.   |
| <i>cfg</i>      | The pointer to the parameters required to configure the LPUART after initialization. |

Returns

0 succeed, others failed

31.5.4.2 `int LPUART_RTOS_Deinit ( lpuart_rtos_handle_t * handle )`

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

|               |                         |
|---------------|-------------------------|
| <i>handle</i> | The RTOS LPUART handle. |
|---------------|-------------------------|

31.5.4.3 `int LPUART_RTOS_Send ( lpuart_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.

## LPUART FreeRTOS Driver

### 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.   |

### 31.5.4.4 int LPUART\_RTOS\_Receive ( lpuart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

### Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS LPUART handle.                                                          |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.                                                  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |

## Chapter 32

# PIT: Periodic Interrupt Timer

### 32.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

### 32.2 Function groups

The PIT driver supports operating the module as a time counter.

#### 32.2.1 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.

#### 32.2.2 Timer period Operations

The function [PITR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function [PIT\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. Users can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds.

#### 32.2.3 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

### 32.2.4 Status

Provides functions to get and clear the PIT status.

### 32.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 32.3 Typical use case

### 32.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pit`

## Data Structures

- struct `pit_config_t`  
*PIT configuration structure. [More...](#)*

## Enumerations

- enum `pit_chnl_t` {  
    `kPIT_Chnl_0` = 0U,  
    `kPIT_Chnl_1`,  
    `kPIT_Chnl_2`,  
    `kPIT_Chnl_3` }  
*List of PIT channels.*
- enum `pit_interrupt_enable_t` { `kPIT_TimerInterruptEnable` = `PIT_TCTRL_TIE_MASK` }  
*List of PIT interrupts.*
- enum `pit_status_flags_t` { `kPIT_TimerFlag` = `PIT_TFLG_TIF_MASK` }  
*List of PIT status flags.*

## Functions

- `uint64_t PIT_GetLifetimeTimerCount` (`PIT_Type *base`)  
*Reads the current lifetime counter value.*

## Driver version

- `#define FSL_PIT_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 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 operations.*
- void `PIT_Deinit` (PIT\_Type \*base)
- *Gates the PIT clock and disables the PIT module.*
- static void `PIT_GetDefaultConfig` (pit\_config\_t \*config)
- *Fills in the PIT configuration structure with the default settings.*
- static void `PIT_SetTimerChainMode` (PIT\_Type \*base, pit\_chnl\_t channel, bool enable)
- *Enables or disables chaining a timer with the previous timer.*

## Interrupt Interface

- static void `PIT_EnableInterrupts` (PIT\_Type \*base, pit\_chnl\_t channel, uint32\_t mask)
- *Enables the selected PIT interrupts.*
- static void `PIT_DisableInterrupts` (PIT\_Type \*base, pit\_chnl\_t channel, uint32\_t mask)
- *Disables the selected PIT interrupts.*
- static uint32\_t `PIT_GetEnabledInterrupts` (PIT\_Type \*base, pit\_chnl\_t channel)
- *Gets the enabled PIT interrupts.*

## Status Interface

- static uint32\_t `PIT_GetStatusFlags` (PIT\_Type \*base, pit\_chnl\_t channel)
- *Gets the PIT status flags.*
- static void `PIT_ClearStatusFlags` (PIT\_Type \*base, pit\_chnl\_t channel, uint32\_t mask)
- *Clears the PIT status flags.*

## Read and Write the timer period

- static void `PIT_SetTimerPeriod` (PIT\_Type \*base, pit\_chnl\_t channel, uint32\_t count)
- *Sets the timer period in units of count.*
- static uint32\_t `PIT_GetCurrentTimerCount` (PIT\_Type \*base, pit\_chnl\_t channel)
- *Reads the current timer counting value.*

## Timer Start and Stop

- static void `PIT_StartTimer` (PIT\_Type \*base, pit\_chnl\_t channel)
- *Starts the timer counting.*
- static void `PIT_StopTimer` (PIT\_Type \*base, pit\_chnl\_t channel)
- *Stops the timer counting.*

## 32.4 Data Structure Documentation

### 32.4.1 struct pit\_config\_t

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the `PIT_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

## Function Documentation

### Data Fields

- bool `enableRunInDebug`  
*true: Timers run in debug mode; false: Timers stop in debug mode*

## 32.5 Enumeration Type Documentation

### 32.5.1 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.

### 32.5.2 enum `pit_interrupt_enable_t`

Enumerator

- kPIT\_TimerInterruptEnable* Timer interrupt enable.

### 32.5.3 enum `pit_status_flags_t`

Enumerator

- kPIT\_TimerFlag* Timer flag.

## 32.6 Function Documentation

### 32.6.1 void `PIT_Init ( PIT_Type * base, const pit_config_t * config )`

Note

This API should be called at the beginning of the application using the PIT driver.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | PIT peripheral base address                |
| <i>config</i> | Pointer to the user's PIT config structure |

### 32.6.2 void PIT\_Deinit ( PIT\_Type \* *base* )

Parameters

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

### 32.6.3 static void PIT\_GetDefaultConfig ( pit\_config\_t \* *config* ) [inline], [static]

The default values are as follows.

```
* config->enableRunInDebug = false;
*
```

Parameters

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

### 32.6.4 static void PIT\_SetTimerChainMode ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, bool *enable* ) [inline], [static]

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

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

## Function Documentation

|                |                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>channel</i> | Timer channel number which is chained with the previous timer                                                                  |
| <i>enable</i>  | Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers. |

### 32.6.5 static void PIT\_EnableInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

#### Parameters

|                |                                                                                                                     |
|----------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                         |
| <i>channel</i> | Timer channel number                                                                                                |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

### 32.6.6 static void PIT\_DisableInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

#### Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | PIT peripheral base address                                                                                          |
| <i>channel</i> | Timer channel number                                                                                                 |
| <i>mask</i>    | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

### 32.6.7 static uint32\_t PIT\_GetEnabledInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel* ) [inline], [static]

#### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

#### Returns

The enabled interrupts. This is the logical OR of members of the enumeration [pit\\_interrupt\\_enable\\_t](#)

**32.6.8** `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](#)

### 32.6.9 static void PIT\_ClearStatusFlags ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

### Parameters

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

### 32.6.10 static void PIT\_SetTimerPeriod ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *count* ) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

### Note

Users can call the utility macros provided in `fsl_common.h` to convert to ticks.

### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

|              |                                |
|--------------|--------------------------------|
| <i>count</i> | Timer period in units of ticks |
|--------------|--------------------------------|

### 32.6.11 `static uint32_t PIT_GetCurrentTimerCount ( PIT_Type * base, pit_chnl_t channel ) [inline], [static]`

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

Users can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec.

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

Current timer counting value in ticks

### 32.6.12 `static void PIT_StartTimer ( PIT_Type * base, pit_chnl_t channel ) [inline], [static]`

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

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

### 32.6.13 `static void PIT_StopTimer ( PIT_Type * base, pit_chnl_t channel ) [inline], [static]`

This function stops every timer counting. Timers reload their periods respectively after the next time they call the `PIT_DRV_StartTimer`.

## Function Documentation

### Parameters

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

### 32.6.14 uint64\_t PIT\_GetLifetimeTimerCount ( PIT\_Type \* *base* )

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the PIT\_SetTimerChainMode before using this timer. The period of lifetime timer is equal to the "period of timer 0 \* period of timer 1". For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

### Parameters

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

### Returns

Current lifetime timer value



## Chapter 33

# PMU: Power Management Unit

### 33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Power Management Unit (PMU) module of MCUXpresso SDK devices. The power management unit (PMU) is designed to simplify the external power interface. The power system can be split into the input power sources and their characteristics, the integrated power transforming and controlling elements, and the final load interconnection and requirements. By using the internal LDO regulators, the number of external supplies is greatly reduced.

The PMU driver provides the APIs to adjust the work condition of each regulator, and can gate the power of some modules.

### Enumerations

- enum `_pmu_status_flags` {  
    `kPMU_1P1RegulatorOutputOK` = (1U << 0U),  
    `kPMU_1P1BrownoutOnOutput` = (1U << 1U),  
    `kPMU_3P0RegulatorOutputOK` = (1U << 2U),  
    `kPMU_3P0BrownoutOnOutput` = (1U << 3U),  
    `kPMU_2P5RegulatorOutputOK` = (1U << 4U),  
    `kPMU_2P5BrownoutOnOutput` = (1U << 5U) }  
*Status flags.*
- enum `pmu_1p1_weak_reference_source_t` {  
    `kPMU_1P1WeakReferenceSourceAlt0` = 0U,  
    `kPMU_1P1WeakReferenceSourceAlt1` = 1U }  
*The source for the reference voltage of the weak 1P1 regulator.*
- enum `pmu_3p0_vbus_voltage_source_t` {  
    `kPMU_3P0VBusVoltageSourceAlt0` = 0U,  
    `kPMU_3P0VBusVoltageSourceAlt1` = 1U }  
*Input voltage source for LDO\_3P0 from USB VBus.*
- enum `pmu_core_reg_voltage_ramp_rate_t` {  
    `kPMU_CoreRegVoltageRampRateFast` = 0U,  
    `kPMU_CoreRegVoltageRampRateMediumFast` = 1U,  
    `kPMU_CoreRegVoltageRampRateMediumSlow` = 2U,  
    `kPMU_CoreRegVoltageRampRateSlow` = 0U }  
*Regulator voltage ramp rate.*
- enum `pmu_power_bandgap_t` {  
    `kPMU_NormalPowerBandgap` = 0U,  
    `kPMU_LowPowerBandgap` = 1U }  
*Bandgap select.*

## Overview

### Driver version

- #define `FSL_PMU_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*PMU driver version.*

### Status.

- `uint32_t PMU_GetStatusFlags` (PMU\_Type \*base)

### 1P1 Regular

- static void `PMU_1P1SetWeakReferenceSource` (PMU\_Type \*base, `pmu_1p1_weak_reference_source_t` option)  
*Selects the source for the reference voltage of the weak 1P1 regulator.*
- static void `PMU_1P1EnableWeakRegulator` (PMU\_Type \*base, bool enable)  
*Enables the weak 1P1 regulator.*
- static void `PMU_1P1SetRegulatorOutputVoltage` (PMU\_Type \*base, `uint32_t` value)  
*Adjust the 1P1 regulator output voltage.*
- static void `PMU_1P1SetBrownoutOffsetVoltage` (PMU\_Type \*base, `uint32_t` value)  
*Adjust the 1P1 regulator brownout offset voltage.*
- static void `PMU_1P1EnablePullDown` (PMU\_Type \*base, bool enable)  
*Enable the pull-down circuitry in the regulator.*
- static void `PMU_1P1EnableCurrentLimit` (PMU\_Type \*base, bool enable)  
*Enable the current-limit circuitry in the regulator.*
- static void `PMU_1P1EnableBrownout` (PMU\_Type \*base, bool enable)  
*Enable the brownout circuitry in the regulator.*
- static void `PMU_1P1EnableOutput` (PMU\_Type \*base, bool enable)  
*Enable the regulator output.*

### 3P0 Regular

- static void `PMU_3P0SetRegulatorOutputVoltage` (PMU\_Type \*base, `uint32_t` value)  
*Adjust the 3P0 regulator output voltage.*
- static void `PMU_3P0SetVBusVoltageSource` (PMU\_Type \*base, `pmu_3p0_vbus_voltage_source_t` option)  
*Select input voltage source for LDO\_3P0.*
- static void `PMU_3P0SetBrownoutOffsetVoltage` (PMU\_Type \*base, `uint32_t` value)  
*Adjust the 3P0 regulator brownout offset voltage.*
- static void `PMU_3P0EnableCurrentLimit` (PMU\_Type \*base, bool enable)  
*Enable the current-limit circuitry in the 3P0 regulator.*
- static void `PMU_3P0EnableBrownout` (PMU\_Type \*base, bool enable)  
*Enable the brownout circuitry in the 3P0 regulator.*
- static void `PMU_3P0EnableOutput` (PMU\_Type \*base, bool enable)  
*Enable the 3P0 regulator output.*

### 2P5 Regulator

- static void `PMU_2P5EnableWeakRegulator` (PMU\_Type \*base, bool enable)  
*Enables the weak 2P5 regulator.*
- static void `PMU_2P5SetRegulatorOutputVoltage` (PMU\_Type \*base, `uint32_t` value)  
*Adjust the 1P1 regulator output voltage.*

- static void [PMU\\_2P5SetBrownoutOffsetVoltage](#) (PMU\_Type \*base, uint32\_t value)  
*Adjust the 2P5 regulator brownout offset voltage.*
- static void [PMU\\_2P5EnablePullDown](#) (PMU\_Type \*base, bool enable)  
*Enable the pull-down circuitry in the 2P5 regulator.*
- static void [PMU\\_2P1EnablePullDown](#) (PMU\_Type \*base, bool enable)  
*Enable the pull-down circuitry in the 2P5 regulator.*
- static void [PMU\\_2P5EnableCurrentLimit](#) (PMU\_Type \*base, bool enable)  
*Enable the current-limit circuitry in the 2P5 regulator.*
- static void [PMU\\_2P5nableBrownout](#) (PMU\_Type \*base, bool enable)  
*Enable the brownout circuitry in the 2P5 regulator.*
- static void [PMU\\_2P5EnableOutput](#) (PMU\_Type \*base, bool enable)  
*Enable the 2P5 regulator output.*

### Core Regulator

- static void [PMU\\_CoreEnableIncreaseGateDrive](#) (PMU\_Type \*base, bool enable)  
*Increase the gate drive on power gating FETs.*
- static void [PMU\\_CoreSetRegulatorVoltageRampRate](#) (PMU\_Type \*base, [pmu\\_core\\_reg\\_voltage\\_ramp\\_rate\\_t](#) option)  
*Set the CORE regulator voltage ramp rate.*
- static void [PMU\\_CoreSetSOCDomainVoltage](#) (PMU\_Type \*base, uint32\_t value)  
*Define the target voltage for the SOC power domain.*
- static void [PMU\\_CoreSetARMCoreDomainVoltage](#) (PMU\_Type \*base, uint32\_t value)  
*Define the target voltage for the ARM Core power domain.*

## 33.2 Macro Definition Documentation

### 33.2.1 #define FSL\_PMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

## 33.3 Enumeration Type Documentation

### 33.3.1 enum \_pmu\_status\_flags

Enumerator

- kPMU\_1P1RegulatorOutputOK*** Status bit that signals when the 1p1 regulator output is ok. 1 = regulator output > brownout target.
- kPMU\_1P1BrownoutOnOutput*** Status bit that signals when a 1p1 brownout is detected on the regulator output.
- kPMU\_3P0RegulatorOutputOK*** Status bit that signals when the 3p0 regulator output is ok. 1 = regulator output > brownout target.
- kPMU\_3P0BrownoutOnOutput*** Status bit that signals when a 3p0 brownout is detected on the regulator output.
- kPMU\_2P5RegulatorOutputOK*** Status bit that signals when the 2p5 regulator output is ok. 1 = regulator output > brownout target.
- kPMU\_2P5BrownoutOnOutput*** Status bit that signals when a 2p5 brownout is detected on the regulator output.

## Function Documentation

### 33.3.2 enum pmu\_1p1\_weak\_reference\_source\_t

Enumerator

- kPMU\_1P1WeakReferenceSourceAlt0* Weak-linreg output tracks low-power-bandgap voltage.
- kPMU\_1P1WeakReferenceSourceAlt1* Weak-linreg output tracks VDD\_SOC\_CAP voltage.

### 33.3.3 enum pmu\_3p0\_vbus\_voltage\_source\_t

Enumerator

- kPMU\_3P0VBusVoltageSourceAlt0* USB\_OTG1\_VBUS - Utilize VBUS OTG1 for power.
- kPMU\_3P0VBusVoltageSourceAlt1* USB\_OTG2\_VBUS - Utilize VBUS OTG2 for power.

### 33.3.4 enum pmu\_core\_reg\_voltage\_ramp\_rate\_t

Enumerator

- kPMU\_CoreRegVoltageRampRateFast* Fast.
- kPMU\_CoreRegVoltageRampRateMediumFast* Medium Fast.
- kPMU\_CoreRegVoltageRampRateMediumSlow* Medium Slow.
- kPMU\_CoreRegVoltageRampRateSlow* Slow.

### 33.3.5 enum pmu\_power\_bandgap\_t

Enumerator

- kPMU\_NormalPowerBandgap* Normal power bandgap.
- kPMU\_LowPowerBandgap* Low power bandgap.

## 33.4 Function Documentation

### 33.4.1 static void PMU\_1P1SetWeakReferenceSource ( PMU\_Type \* base, pmu\_1p1\_weak\_reference\_source\_t option ) [inline], [static]

Parameters

---

|               |                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>   | PMU peripheral base address.                                                                      |
| <i>option</i> | The option for reference voltage source, see to <a href="#">pmu_1p1_weak_reference_source_t</a> . |

### 33.4.2 static void PMU\_1P1EnableWeakRegulator ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]

This regulator can be used when the main 1P1 regulator is disabled, under low-power conditions.

Parameters

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

### 33.4.3 static void PMU\_1P1SetRegulatorOutputVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x1b(1.375V) >= output\_trg >= 0x04(0.8V)
- 0x04 : 0.8V
- 0x10 : 1.1V (typical)
- 0x1b : 1.375V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | PMU peripheral base address.  |
| <i>value</i> | Setting value for the output. |

### 33.4.4 static void PMU\_1P1SetBrownoutOffsetVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Control bits to adjust the regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT\_TRG - BO\_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

## Function Documentation

Parameters

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                                            |
| <i>value</i> | Setting value for the brownout offset. The available range is in 3-bit. |

**33.4.5 static void PMU\_1P1EnablePullDown ( PMU\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

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

**33.4.6 static void PMU\_1P1EnableCurrentLimit ( PMU\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

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

**33.4.7 static void PMU\_1P1EnableBrownout ( PMU\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

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

**33.4.8 static void PMU\_1P1EnableOutput ( PMU\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

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

### 33.4.9 static void PMU\_3P0SetRegulatorOutputVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.625V) >= output\_trg >= 0x1f(3.4V)
- 0x00 : 2.625V
- 0x0f : 3.0V (typical)
- 0x1f : 3.4V

Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | PMU peripheral base address.  |
| <i>value</i> | Setting value for the output. |

### 33.4.10 static void PMU\_3P0SetVBusVoltageSource ( PMU\_Type \* *base*, pmu\_3p0\_vbus\_voltage\_source\_t *option* ) [inline], [static]

Select input voltage source for LDO\_3P0 from either USB\_OTG1\_VBUS or USB\_OTG2\_VBUS. If only one of the two VBUS voltages is present, it is automatically selected.

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | PMU peripheral base address.                   |
| <i>option</i> | User-defined input voltage source for LDO_3P0. |

### 33.4.11 static void PMU\_3P0SetBrownoutOffsetVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Control bits to adjust the 3P0 regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT\_TRG - BO\_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

## Function Documentation

### Parameters

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                                            |
| <i>value</i> | Setting value for the brownout offset. The available range is in 3-bit. |

**33.4.12** `static void PMU_3P0EnableCurrentLimit ( PMU_Type * base, bool enable )`  
`[inline], [static]`

### Parameters

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

**33.4.13** `static void PMU_3P0EnableBrownout ( PMU_Type * base, bool enable )`  
`[inline], [static]`

### Parameters

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

**33.4.14** `static void PMU_3P0EnableOutput ( PMU_Type * base, bool enable )`  
`[inline], [static]`

### Parameters

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

**33.4.15** `static void PMU_2P5EnableWeakRegulator ( PMU_Type * base, bool enable )`  
`[inline], [static]`

This low power regulator is used when the main 2P5 regulator is disabled to keep the 2.5V output roughly at 2.5V. Scales directly with the value of VDDHIGH\_IN.



## Parameters

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

### 33.4.16 static void PMU\_2P5SetRegulatorOutputVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Each LSB is worth 25mV. Programming examples are detailed below. Other output target voltages may be interpolated from these examples. Choices must be in this range:

- 0x00(2.1V) >= output\_trg >= 0x1f(2.875V)
- 0x00 : 2.1V
- 0x10 : 2.5V (typical)
- 0x1f : 2.875V NOTE: There may be reduced chip functionality or reliability at the extremes of the programming range.

## Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | PMU peripheral base address.  |
| <i>value</i> | Setting value for the output. |

### 33.4.17 static void PMU\_2P5SetBrownoutOffsetVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Adjust the regulator brownout offset voltage in 25mV steps. The reset brown-offset is 175mV below the programmed target code. Brownout target = OUTPUT\_TRG - BO\_OFFSET. Some steps may be irrelevant because of input supply limitations or load operation.

## Parameters

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                                            |
| <i>value</i> | Setting value for the brownout offset. The available range is in 3-bit. |

### 33.4.18 static void PMU\_2P5EnablePullDown ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]

## Function Documentation

Parameters

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

**33.4.19** `static void PMU_2P1EnablePullDown ( PMU_Type * base, bool enable )`  
`[inline], [static]`

**33.4.20** `static void PMU_2P5EnableCurrentLimit ( PMU_Type * base, bool enable )`  
`[inline], [static]`

Parameters

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

**33.4.21** `static void PMU_2P5nableBrownout ( PMU_Type * base, bool enable )`  
`[inline], [static]`

Parameters

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

**33.4.22** `static void PMU_2P5EnableOutput ( PMU_Type * base, bool enable )`  
`[inline], [static]`

Parameters

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

### 33.4.23 `static void PMU_CoreEnableIncreaseGateDrive ( PMU_Type * base, bool enable ) [inline], [static]`

If set, increases the gate drive on power gating FETs to reduce leakage in the off state. Care must be taken to apply this bit only when the input supply voltage to the power FET is less than 1.1V. NOTE: This bit should only be used in low-power modes where the external input supply voltage is nominally 0.9V.

Parameters

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

### 33.4.24 `static void PMU_CoreSetRegulatorVoltageRampRate ( PMU_Type * base, pmu_core_reg_voltage_ramp_rate_t option ) [inline], [static]`

Parameters

|               |                                                                                                      |
|---------------|------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PMU peripheral base address.                                                                         |
| <i>option</i> | User-defined option for voltage ramp rate, see to <a href="#">pmu_core_reg_voltage_ramp_rate_t</a> . |

### 33.4.25 `static void PMU_CoreSetSOCDomainVoltage ( PMU_Type * base, uint32_t value ) [inline], [static]`

Define the target voltage for the SOC power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V
- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

## Function Documentation

Parameters

|              |                                                   |
|--------------|---------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                      |
| <i>value</i> | Setting value for target voltage. 5-bit available |

### 33.4.26 static void PMU\_CoreSetARMCoreDomainVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Define the target voltage for the ARM Core power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V
- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

Parameters

|              |                                                   |
|--------------|---------------------------------------------------|
| <i>base</i>  | PMU peripheral base address.                      |
| <i>value</i> | Setting value for target voltage. 5-bit available |

## Chapter 34

# PWM: Pulse Width Modulator

### 34.1 Overview

The MCUXpresso SDK provides a driver for the Pulse Width Modulator (PWM) of MCUXpresso SDK devices.

The function `PWM_Init()` initializes the PWM sub module with specified configurations, the function `PWM_GetDefaultConfig()` could help to get the default configurations. The initialization function configures the sub module for the requested register update mode for registers with buffers. It also sets up the sub module operation in debug and wait modes.

The function `PWM_SetupPwm()` sets up PWM channels for PWM output, the function can set up PWM signal properties for multiple channels. The PWM has 2 channels: A and B. Each channel has its own duty cycle and level-mode specified, however the same PWM period and PWM mode is applied to all channels requesting PWM output. The signal duty cycle is provided as a percentage of the PWM period, its value should be between 0 and 100; 0=inactive signal(0% duty cycle) and 100=always active signal (100% duty cycle). The function also sets up the channel dead time value which is used when the user selects complementary mode of operation.

The function `PWM_UpdatePwmDutycycle()` updates the PWM signal duty cycle of a particular PWM channel.

The function `PWM_SetupInputCapture()` sets up a PWM channel for input capture. The user can specify the capture edge and the mode; one-shot capture or free-running capture.

The function `PWM_SetupFault()` sets up the properties for each fault.

The function `PWM_StartTimer()` can be used to start one or multiple sub modules. The function `PWM_StopTimer()` can be used to stop one or multiple sub modules.

Provide functions to get and clear the PWM status.

Provide functions to enable/disable PWM interrupts and get current enabled interrupts.

### 34.2 Register Update

Some of the PWM registers have buffers, the driver support various methods to update these registers with the content of the register buffer. The update mechanism for register with buffers can be specified through the following fields available in the configuration structure. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pwm`The user can select one of the reload options provided in enumeration `pwm_register_reload_t`. When using immediate reload, the `reloadFrequency` field is not used.

The driver initialization function sets up the appropriate bits in the PWM module based on the register update options selected.

The below function should be used to initiate a register reload. The example shows register reload ini-

## Typical use case

tiated on PWM sub modules 0, 1, and 2. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pwm

## 34.3 Typical use case

### 34.3.1 PWM output

Output PWM signal on 3 PWM sub module with different dutycycles. Periodically update the PWM signal duty cycle. Each sub module runs in Complementary output mode with PWM A used to generate the complementary PWM pair. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pwm

## Data Structures

- struct [pwm\\_signal\\_param\\_t](#)  
*Structure for the user to define the PWM signal characteristics. [More...](#)*
- struct [pwm\\_config\\_t](#)  
*PWM config structure. [More...](#)*
- struct [pwm\\_fault\\_param\\_t](#)  
*Structure is used to hold the parameters to configure a PWM fault. [More...](#)*
- struct [pwm\\_input\\_capture\\_param\\_t](#)  
*Structure is used to hold parameters to configure the capture capability of a signal pin. [More...](#)*

## Macros

- #define [PWM\\_SUBMODULE\\_SWCONTROL\\_WIDTH](#) 2  
*Number of bits per submodule for software output control.*

## Enumerations

- enum [pwm\\_submodule\\_t](#) {  
  [kPWM\\_Module\\_0](#) = 0U,  
  [kPWM\\_Module\\_1](#),  
  [kPWM\\_Module\\_2](#),  
  [kPWM\\_Module\\_3](#) }  
*List of PWM submodules.*
- enum [pwm\\_channels\\_t](#)  
*List of PWM channels in each module.*
- enum [pwm\\_value\\_register\\_t](#) {  
  [kPWM\\_ValueRegister\\_0](#) = 0U,  
  [kPWM\\_ValueRegister\\_1](#),  
  [kPWM\\_ValueRegister\\_2](#),  
  [kPWM\\_ValueRegister\\_3](#),  
  [kPWM\\_ValueRegister\\_4](#),  
  [kPWM\\_ValueRegister\\_5](#) }  
*List of PWM value registers.*

- enum `pwm_clock_source_t` {  
`kPWM_BusClock = 0U,`  
`kPWM_ExternalClock,`  
`kPWM_Submodule0Clock }`  
*PWM clock source selection.*
- enum `pwm_clock_prescale_t` {  
`kPWM_Prescale_Divide_1 = 0U,`  
`kPWM_Prescale_Divide_2,`  
`kPWM_Prescale_Divide_4,`  
`kPWM_Prescale_Divide_8,`  
`kPWM_Prescale_Divide_16,`  
`kPWM_Prescale_Divide_32,`  
`kPWM_Prescale_Divide_64,`  
`kPWM_Prescale_Divide_128 }`  
*PWM prescaler factor selection for clock source.*
- enum `pwm_force_output_trigger_t` {  
`kPWM_Force_Local = 0U,`  
`kPWM_Force_Master,`  
`kPWM_Force_LocalReload,`  
`kPWM_Force_MasterReload,`  
`kPWM_Force_LocalSync,`  
`kPWM_Force_MasterSync,`  
`kPWM_Force_External,`  
`kPWM_Force_ExternalSync }`  
*Options that can trigger a PWM FORCE\_OUT.*
- enum `pwm_init_source_t` {  
`kPWM_Initialize_LocalSync = 0U,`  
`kPWM_Initialize_MasterReload,`  
`kPWM_Initialize_MasterSync,`  
`kPWM_Initialize_ExtSync }`  
*PWM counter initialization options.*
- enum `pwm_load_frequency_t` {

## Typical use case

```
kPWM_LoadEveryOportunity = 0U,  
kPWM_LoadEvery2Oportunity,  
kPWM_LoadEvery3Oportunity,  
kPWM_LoadEvery4Oportunity,  
kPWM_LoadEvery5Oportunity,  
kPWM_LoadEvery6Oportunity,  
kPWM_LoadEvery7Oportunity,  
kPWM_LoadEvery8Oportunity,  
kPWM_LoadEvery9Oportunity,  
kPWM_LoadEvery10Oportunity,  
kPWM_LoadEvery11Oportunity,  
kPWM_LoadEvery12Oportunity,  
kPWM_LoadEvery13Oportunity,  
kPWM_LoadEvery14Oportunity,  
kPWM_LoadEvery15Oportunity,  
kPWM_LoadEvery16Oportunity }
```

*PWM load frequency selection.*

- enum `pwm_fault_input_t` {  
kPWM\_Fault\_0 = 0U,  
kPWM\_Fault\_1,  
kPWM\_Fault\_2,  
kPWM\_Fault\_3 }

*List of PWM fault selections.*

- enum `pwm_input_capture_edge_t` {  
kPWM\_Disable = 0U,  
kPWM\_FallingEdge,  
kPWM\_RisingEdge,  
kPWM\_RiseAndFallEdge }

*PWM capture edge select.*

- enum `pwm_force_signal_t` {  
kPWM\_UsePwm = 0U,  
kPWM\_InvertedPwm,  
kPWM\_SoftwareControl,  
kPWM\_UseExternal }

*PWM output options when a FORCE\_OUT signal is asserted.*

- enum `pwm_chnl_pair_operation_t` {  
kPWM\_Independent = 0U,  
kPWM\_ComplementaryPwmA,  
kPWM\_ComplementaryPwmB }

*Options available for the PWM A & B pair operation.*

- enum `pwm_register_reload_t` {  
kPWM\_ReloadImmediate = 0U,  
kPWM\_ReloadPwmHalfCycle,  
kPWM\_ReloadPwmFullCycle,  
kPWM\_ReloadPwmHalfAndFullCycle }

*Options available on how to load the buffered-registers with new values.*



- enum `pwm_fault_recovery_mode_t` {  
`kPWM_NoRecovery = 0U`,  
`kPWM_RecoverHalfCycle`,  
`kPWM_RecoverFullCycle`,  
`kPWM_RecoverHalfAndFullCycle` }

*Options available on how to re-enable the PWM output when recovering from a fault.*

- enum `pwm_interrupt_enable_t` {  
`kPWM_CompareVal0InterruptEnable = (1U << 0)`,  
`kPWM_CompareVal1InterruptEnable = (1U << 1)`,  
`kPWM_CompareVal2InterruptEnable = (1U << 2)`,  
`kPWM_CompareVal3InterruptEnable = (1U << 3)`,  
`kPWM_CompareVal4InterruptEnable = (1U << 4)`,  
`kPWM_CompareVal5InterruptEnable = (1U << 5)`,  
`kPWM_CaptureX0InterruptEnable = (1U << 6)`,  
`kPWM_CaptureX1InterruptEnable = (1U << 7)`,  
`kPWM_CaptureB0InterruptEnable = (1U << 8)`,  
`kPWM_CaptureB1InterruptEnable = (1U << 9)`,  
`kPWM_CaptureA0InterruptEnable = (1U << 10)`,  
`kPWM_CaptureA1InterruptEnable = (1U << 11)`,  
`kPWM_ReloadInterruptEnable = (1U << 12)`,  
`kPWM_ReloadErrorInterruptEnable = (1U << 13)`,  
`kPWM_Fault0InterruptEnable = (1U << 16)`,  
`kPWM_Fault1InterruptEnable = (1U << 17)`,  
`kPWM_Fault2InterruptEnable = (1U << 18)`,  
`kPWM_Fault3InterruptEnable = (1U << 19)` }

*List of PWM interrupt options.*

- enum `pwm_status_flags_t` {  
`kPWM_CompareVal0Flag = (1U << 0)`,  
`kPWM_CompareVal1Flag = (1U << 1)`,  
`kPWM_CompareVal2Flag = (1U << 2)`,  
`kPWM_CompareVal3Flag = (1U << 3)`,  
`kPWM_CompareVal4Flag = (1U << 4)`,  
`kPWM_CompareVal5Flag = (1U << 5)`,  
`kPWM_CaptureX0Flag = (1U << 6)`,  
`kPWM_CaptureX1Flag = (1U << 7)`,  
`kPWM_CaptureB0Flag = (1U << 8)`,  
`kPWM_CaptureB1Flag = (1U << 9)`,  
`kPWM_CaptureA0Flag = (1U << 10)`,  
`kPWM_CaptureA1Flag = (1U << 11)`,  
`kPWM_ReloadFlag = (1U << 12)`,  
`kPWM_ReloadErrorFlag = (1U << 13)`,  
`kPWM_RegUpdatedFlag = (1U << 14)`,  
`kPWM_Fault0Flag = (1U << 16)`,  
`kPWM_Fault1Flag = (1U << 17)`,  
`kPWM_Fault2Flag = (1U << 18)`,

## Typical use case

```
kPWM_Fault3Flag = (1U << 19) }
```

*List of PWM status flags.*

- enum `pwm_mode_t` {  
    `kPWM_SignedCenterAligned` = 0U,  
    `kPWM_CenterAligned`,  
    `kPWM_SignedEdgeAligned`,  
    `kPWM_EdgeAligned` }  
*PWM operation mode.*
- enum `pwm_level_select_t` {  
    `kPWM_HighTrue` = 0U,  
    `kPWM_LowTrue` }  
*PWM output pulse mode, high-true or low-true.*
- enum `pwm_reload_source_select_t` {  
    `kPWM_LocalReload` = 0U,  
    `kPWM_MasterReload` }  
*PWM reload source select.*
- enum `pwm_fault_clear_t` {  
    `kPWM_Automatic` = 0U,  
    `kPWM_ManualNormal`,  
    `kPWM_ManualSafety` }  
*PWM fault clearing options.*
- enum `pwm_module_control_t` {  
    `kPWM_Control_Module_0` = (1U << 0),  
    `kPWM_Control_Module_1` = (1U << 1),  
    `kPWM_Control_Module_2` = (1U << 2),  
    `kPWM_Control_Module_3` = (1U << 3) }  
*Options for submodule master control operation.*

## Functions

- void `PWM_SetupInputCapture` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_channels_t pwmChannel`, const `pwm_input_capture_param_t *inputCaptureParams`)  
*Sets up the PWM input capture.*
- void `PWM_SetupFaults` (`PWM_Type *base`, `pwm_fault_input_t faultNum`, const `pwm_fault_param_t *faultParams`)  
*Sets up the PWM fault protection.*
- void `PWM_SetupForceSignal` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_channels_t pwmChannel`, `pwm_force_signal_t mode`)  
*Selects the signal to output on a PWM pin when a FORCE\_OUT signal is asserted.*
- static void `PWM_OutputTriggerEnable` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_value_register_t valueRegister`, bool activate)  
*Enables or disables the PWM output trigger.*
- static void `PWM_SetupSwCtrlOut` (`PWM_Type *base`, `pwm_submodule_t subModule`, `pwm_channels_t pwmChannel`, bool value)  
*Sets the software control output for a pin to high or low.*
- static void `PWM_SetPwmLdok` (`PWM_Type *base`, `uint8_t subModulesToUpdate`, bool value)  
*Sets or clears the PWM LDOK bit on a single or multiple submodules.*

## Driver version

- #define `FSL_PWM_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- status\_t `PWM_Init` (PWM\_Type \*base, `pwm_submodule_t` subModule, const `pwm_config_t` \*config)  
*Ungates the PWM submodule clock and configures the peripheral for basic operation.*
- void `PWM_Deinit` (PWM\_Type \*base, `pwm_submodule_t` subModule)  
*Gate the PWM submodule clock.*
- void `PWM_GetDefaultConfig` (`pwm_config_t` \*config)  
*Fill in the PWM config struct with the default settings.*

## Module PWM output

- status\_t `PWM_SetupPwm` (PWM\_Type \*base, `pwm_submodule_t` subModule, const `pwm_signal_param_t` \*chnlParams, uint8\_t numOfChnls, `pwm_mode_t` mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)  
*Sets up the PWM signals for a PWM submodule.*
- void `PWM_UpdatePwmDutycycle` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmSignal, `pwm_mode_t` currPwmMode, uint8\_t dutyCyclePercent)  
*Updates the PWM signal's dutycycle.*

## Interrupts Interface

- void `PWM_EnableInterrupts` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint32\_t mask)  
*Enables the selected PWM interrupts.*
- void `PWM_DisableInterrupts` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint32\_t mask)  
*Disables the selected PWM interrupts.*
- uint32\_t `PWM_GetEnabledInterrupts` (PWM\_Type \*base, `pwm_submodule_t` subModule)  
*Gets the enabled PWM interrupts.*

## Status Interface

- uint32\_t `PWM_GetStatusFlags` (PWM\_Type \*base, `pwm_submodule_t` subModule)  
*Gets the PWM status flags.*
- void `PWM_ClearStatusFlags` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint32\_t mask)  
*Clears the PWM status flags.*

## Timer Start and Stop

- static void `PWM_StartTimer` (PWM\_Type \*base, uint8\_t subModulesToStart)  
*Starts the PWM counter for a single or multiple submodules.*
- static void `PWM_StopTimer` (PWM\_Type \*base, uint8\_t subModulesToStop)  
*Stops the PWM counter for a single or multiple submodules.*

### 34.4 Data Structure Documentation

#### 34.4.1 struct pwm\_signal\_param\_t

##### Data Fields

- [pwm\\_channels\\_t pwmChannel](#)  
*PWM channel being configured; PWM A or PWM B.*
- [uint8\\_t dutyCyclePercent](#)  
*PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*
- [pwm\\_level\\_select\\_t level](#)  
*PWM output active level select.*
- [uint16\\_t deadtimeValue](#)  
*The deadtime value; only used if channel pair is operating in complementary mode.*

##### 34.4.1.0.0.29 Field Documentation

###### 34.4.1.0.0.29.1 uint8\_t pwm\_signal\_param\_t::dutyCyclePercent

100=always active signal (100% duty cycle)

#### 34.4.2 struct pwm\_config\_t

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the [PWM\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

##### Data Fields

- [bool enableDebugMode](#)  
*true: PWM continues to run in debug mode; false: PWM is paused in debug mode*
- [bool enableWait](#)  
*true: PWM continues to run in WAIT mode; false: PWM is paused in WAIT mode*
- [uint8\\_t faultFilterCount](#)  
*Fault filter count.*
- [uint8\\_t faultFilterPeriod](#)  
*Fault filter period;value of 0 will bypass the filter.*
- [pwm\\_init\\_source\\_t initializationControl](#)  
*Option to initialize the counter.*
- [pwm\\_clock\\_source\\_t clockSource](#)  
*Clock source for the counter.*
- [pwm\\_clock\\_prescale\\_t prescale](#)  
*Pre-scaler to divide down the clock.*
- [pwm\\_chnl\\_pair\\_operation\\_t pairOperation](#)  
*Channel pair in independent or complementary mode.*
- [pwm\\_register\\_reload\\_t reloadLogic](#)

- *PWM Reload logic setup.*
- [pwm\\_reload\\_source\\_select\\_t reloadSelect](#)  
*Reload source select.*
- [pwm\\_load\\_frequency\\_t reloadFrequency](#)  
*Specifies when to reload, used when user's choice is not immediate reload.*
- [pwm\\_force\\_output\\_trigger\\_t forceTrigger](#)  
*Specify which signal will trigger a FORCE\_OUT.*

### 34.4.3 struct pwm\_fault\_param\_t

#### Data Fields

- [pwm\\_fault\\_clear\\_t faultClearingMode](#)  
*Fault clearing mode to use.*
- bool [faultLevel](#)  
*true: Logic 1 indicates fault; false: Logic 0 indicates fault*
- bool [enableCombinationalPath](#)  
*true: Combinational Path from fault input is enabled; false: No combination path is available*
- [pwm\\_fault\\_recovery\\_mode\\_t recoverMode](#)  
*Specify when to re-enable the PWM output.*

### 34.4.4 struct pwm\_input\_capture\_param\_t

#### Data Fields

- bool [captureInputSel](#)  
*true: Use the edge counter signal as source false: Use the raw input signal from the pin as source*
- [uint8\\_t edgeCompareValue](#)  
*Compare value, used only if edge counter is used as source.*
- [pwm\\_input\\_capture\\_edge\\_t edge0](#)  
*Specify which edge causes a capture for input circuitry 0.*
- [pwm\\_input\\_capture\\_edge\\_t edge1](#)  
*Specify which edge causes a capture for input circuitry 1.*
- bool [enableOneShotCapture](#)  
*true: Use one-shot capture mode; false: Use free-running capture mode*
- [uint8\\_t fifoWatermark](#)  
*Watermark level for capture FIFO.*

#### 34.4.4.0.0.30 Field Documentation

##### 34.4.4.0.0.30.1 uint8\_t pwm\_input\_capture\_param\_t::fifoWatermark

The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

## Enumeration Type Documentation

### 34.5 Enumeration Type Documentation

#### 34.5.1 enum pwm\_submodule\_t

Enumerator

*kPWM\_Module\_0* Submodule 0.  
*kPWM\_Module\_1* Submodule 1.  
*kPWM\_Module\_2* Submodule 2.  
*kPWM\_Module\_3* Submodule 3.

#### 34.5.2 enum pwm\_value\_register\_t

Enumerator

*kPWM\_ValueRegister\_0* PWM Value0 register.  
*kPWM\_ValueRegister\_1* PWM Value1 register.  
*kPWM\_ValueRegister\_2* PWM Value2 register.  
*kPWM\_ValueRegister\_3* PWM Value3 register.  
*kPWM\_ValueRegister\_4* PWM Value4 register.  
*kPWM\_ValueRegister\_5* PWM Value5 register.

#### 34.5.3 enum pwm\_clock\_source\_t

Enumerator

*kPWM\_BusClock* The IPBus clock is used as the clock.  
*kPWM\_ExternalClock* EXT\_CLK is used as the clock.  
*kPWM\_Submodule0Clock* Clock of the submodule 0 (AUX\_CLK) is used as the source clock.

#### 34.5.4 enum pwm\_clock\_prescale\_t

Enumerator

*kPWM\_Prescale\_Divide\_1* PWM clock frequency = fclk/1.  
*kPWM\_Prescale\_Divide\_2* PWM clock frequency = fclk/2.  
*kPWM\_Prescale\_Divide\_4* PWM clock frequency = fclk/4.  
*kPWM\_Prescale\_Divide\_8* PWM clock frequency = fclk/8.  
*kPWM\_Prescale\_Divide\_16* PWM clock frequency = fclk/16.  
*kPWM\_Prescale\_Divide\_32* PWM clock frequency = fclk/32.  
*kPWM\_Prescale\_Divide\_64* PWM clock frequency = fclk/64.  
*kPWM\_Prescale\_Divide\_128* PWM clock frequency = fclk/128.

### 34.5.5 enum pwm\_force\_output\_trigger\_t

Enumerator

- kPWM\_Force\_Local* The local force signal, CTRL2[FORCE], from the submodule is used to force updates.
- kPWM\_Force\_Master* The master force signal from submodule 0 is used to force updates.
- kPWM\_Force\_LocalReload* The local reload signal from this submodule is used to force updates without regard to the state of LDOK.
- kPWM\_Force\_MasterReload* The master reload signal from submodule 0 is used to force updates if LDOK is set.
- kPWM\_Force\_LocalSync* The local sync signal from this submodule is used to force updates.
- kPWM\_Force\_MasterSync* The master sync signal from submodule0 is used to force updates.
- kPWM\_Force\_External* The external force signal, EXT\_FORCE, from outside the PWM module causes updates.
- kPWM\_Force\_ExternalSync* The external sync signal, EXT\_SYNC, from outside the PWM module causes updates.

### 34.5.6 enum pwm\_init\_source\_t

Enumerator

- kPWM\_Initialize\_LocalSync* Local sync causes initialization.
- kPWM\_Initialize\_MasterReload* Master reload from submodule 0 causes initialization.
- kPWM\_Initialize\_MasterSync* Master sync from submodule 0 causes initialization.
- kPWM\_Initialize\_ExtSync* EXT\_SYNC causes initialization.

### 34.5.7 enum pwm\_load\_frequency\_t

Enumerator

- kPWM\_LoadEveryOpportunity* Every PWM opportunity.
- kPWM\_LoadEvery2Opportunity* Every 2 PWM opportunities.
- kPWM\_LoadEvery3Opportunity* Every 3 PWM opportunities.
- kPWM\_LoadEvery4Opportunity* Every 4 PWM opportunities.
- kPWM\_LoadEvery5Opportunity* Every 5 PWM opportunities.
- kPWM\_LoadEvery6Opportunity* Every 6 PWM opportunities.
- kPWM\_LoadEvery7Opportunity* Every 7 PWM opportunities.
- kPWM\_LoadEvery8Opportunity* Every 8 PWM opportunities.
- kPWM\_LoadEvery9Opportunity* Every 9 PWM opportunities.
- kPWM\_LoadEvery10Opportunity* Every 10 PWM opportunities.
- kPWM\_LoadEvery11Opportunity* Every 11 PWM opportunities.
- kPWM\_LoadEvery12Opportunity* Every 12 PWM opportunities.

## Enumeration Type Documentation

- kPWM\_LoadEvery13Opportunity* Every 13 PWM opportunities.
- kPWM\_LoadEvery14Opportunity* Every 14 PWM opportunities.
- kPWM\_LoadEvery15Opportunity* Every 15 PWM opportunities.
- kPWM\_LoadEvery16Opportunity* Every 16 PWM opportunities.

### 34.5.8 enum pwm\_fault\_input\_t

Enumerator

- kPWM\_Fault\_0* Fault 0 input pin.
- kPWM\_Fault\_1* Fault 1 input pin.
- kPWM\_Fault\_2* Fault 2 input pin.
- kPWM\_Fault\_3* Fault 3 input pin.

### 34.5.9 enum pwm\_input\_capture\_edge\_t

Enumerator

- kPWM\_Disable* Disabled.
- kPWM\_FallingEdge* Capture on falling edge only.
- kPWM\_RisingEdge* Capture on rising edge only.
- kPWM\_RiseAndFallEdge* Capture on rising or falling edge.

### 34.5.10 enum pwm\_force\_signal\_t

Enumerator

- kPWM\_UsePwm* Generated PWM signal is used by the deadtime logic.
- kPWM\_InvertedPwm* Inverted PWM signal is used by the deadtime logic.
- kPWM\_SoftwareControl* Software controlled value is used by the deadtime logic.
- kPWM\_UseExternal* PWM\_EXT\_A signal is used by the deadtime logic.

### 34.5.11 enum pwm\_chnl\_pair\_operation\_t

Enumerator

- kPWM\_Independent* PWM A & PWM B operate as 2 independent channels.
- kPWM\_ComplementaryPwmA* PWM A & PWM B are complementary channels, PWM A generates the signal.
- kPWM\_ComplementaryPwmB* PWM A & PWM B are complementary channels, PWM B generates the signal.



### 34.5.12 enum pwm\_register\_reload\_t

Enumerator

*kPWM\_ReloadImmediate* Buffered-registers get loaded with new values as soon as LDOK bit is set.

*kPWM\_ReloadPwmHalfCycle* Registers loaded on a PWM half cycle.

*kPWM\_ReloadPwmFullCycle* Registers loaded on a PWM full cycle.

*kPWM\_ReloadPwmHalfAndFullCycle* Registers loaded on a PWM half & full cycle.

### 34.5.13 enum pwm\_fault\_recovery\_mode\_t

Enumerator

*kPWM\_NoRecovery* PWM output will stay inactive.

*kPWM\_RecoverHalfCycle* PWM output re-enabled at the first half cycle.

*kPWM\_RecoverFullCycle* PWM output re-enabled at the first full cycle.

*kPWM\_RecoverHalfAndFullCycle* PWM output re-enabled at the first half or full cycle.

### 34.5.14 enum pwm\_interrupt\_enable\_t

Enumerator

*kPWM\_CompareVal0InterruptEnable* PWM VAL0 compare interrupt.

*kPWM\_CompareVal1InterruptEnable* PWM VAL1 compare interrupt.

*kPWM\_CompareVal2InterruptEnable* PWM VAL2 compare interrupt.

*kPWM\_CompareVal3InterruptEnable* PWM VAL3 compare interrupt.

*kPWM\_CompareVal4InterruptEnable* PWM VAL4 compare interrupt.

*kPWM\_CompareVal5InterruptEnable* PWM VAL5 compare interrupt.

*kPWM\_CaptureX0InterruptEnable* PWM capture X0 interrupt.

*kPWM\_CaptureX1InterruptEnable* PWM capture X1 interrupt.

*kPWM\_CaptureB0InterruptEnable* PWM capture B0 interrupt.

*kPWM\_CaptureB1InterruptEnable* PWM capture B1 interrupt.

*kPWM\_CaptureA0InterruptEnable* PWM capture A0 interrupt.

*kPWM\_CaptureA1InterruptEnable* PWM capture A1 interrupt.

*kPWM\_ReloadInterruptEnable* PWM reload interrupt.

*kPWM\_ReloadErrorInterruptEnable* PWM reload error interrupt.

*kPWM\_Fault0InterruptEnable* PWM fault 0 interrupt.

*kPWM\_Fault1InterruptEnable* PWM fault 1 interrupt.

*kPWM\_Fault2InterruptEnable* PWM fault 2 interrupt.

*kPWM\_Fault3InterruptEnable* PWM fault 3 interrupt.

## Enumeration Type Documentation

### 34.5.15 enum pwm\_status\_flags\_t

Enumerator

*kPWM\_CompareVal0Flag* PWM VAL0 compare flag.  
*kPWM\_CompareVal1Flag* PWM VAL1 compare flag.  
*kPWM\_CompareVal2Flag* PWM VAL2 compare flag.  
*kPWM\_CompareVal3Flag* PWM VAL3 compare flag.  
*kPWM\_CompareVal4Flag* PWM VAL4 compare flag.  
*kPWM\_CompareVal5Flag* PWM VAL5 compare flag.  
*kPWM\_CaptureX0Flag* PWM capture X0 flag.  
*kPWM\_CaptureX1Flag* PWM capture X1 flag.  
*kPWM\_CaptureB0Flag* PWM capture B0 flag.  
*kPWM\_CaptureB1Flag* PWM capture B1 flag.  
*kPWM\_CaptureA0Flag* PWM capture A0 flag.  
*kPWM\_CaptureA1Flag* PWM capture A1 flag.  
*kPWM\_ReloadFlag* PWM reload flag.  
*kPWM\_ReloadErrorFlag* PWM reload error flag.  
*kPWM\_RegUpdatedFlag* PWM registers updated flag.  
*kPWM\_Fault0Flag* PWM fault 0 flag.  
*kPWM\_Fault1Flag* PWM fault 1 flag.  
*kPWM\_Fault2Flag* PWM fault 2 flag.  
*kPWM\_Fault3Flag* PWM fault 3 flag.

### 34.5.16 enum pwm\_mode\_t

Enumerator

*kPWM\_SignedCenterAligned* Signed center-aligned.  
*kPWM\_CenterAligned* Unsigned center-aligned.  
*kPWM\_SignedEdgeAligned* Signed edge-aligned.  
*kPWM\_EdgeAligned* Unsigned edge-aligned.

### 34.5.17 enum pwm\_level\_select\_t

Enumerator

*kPWM\_HighTrue* High level represents "on" or "active" state.  
*kPWM\_LowTrue* Low level represents "on" or "active" state.

### 34.5.18 enum pwm\_reload\_source\_select\_t

Enumerator

*kPWM\_LocalReload* The local reload signal is used to reload registers.

*kPWM\_MasterReload* The master reload signal (from submodule 0) is used to reload.

### 34.5.19 enum pwm\_fault\_clear\_t

Enumerator

*kPWM\_Automatic* Automatic fault clearing.

*kPWM\_ManualNormal* Manual fault clearing with no fault safety mode.

*kPWM\_ManualSafety* Manual fault clearing with fault safety mode.

### 34.5.20 enum pwm\_module\_control\_t

Enumerator

*kPWM\_Control\_Module\_0* Control submodule 0's start/stop,buffer reload operation.

*kPWM\_Control\_Module\_1* Control submodule 1's start/stop,buffer reload operation.

*kPWM\_Control\_Module\_2* Control submodule 2's start/stop,buffer reload operation.

*kPWM\_Control\_Module\_3* Control submodule 3's start/stop,buffer reload operation.

## 34.6 Function Documentation

### 34.6.1 status\_t PWM\_Init ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, const pwm\_config\_t \* *config* )

Note

This API should be called at the beginning of the application using the PWM driver.

Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

## Function Documentation

|               |                                         |
|---------------|-----------------------------------------|
| <i>config</i> | Pointer to user's PWM config structure. |
|---------------|-----------------------------------------|

Returns

kStatus\_Success means success; else failed.

### 34.6.2 void PWM\_Deinit ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule* )

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | PWM peripheral base address   |
| <i>subModule</i> | PWM submodule to deinitialize |

### 34.6.3 void PWM\_GetDefaultConfig ( pwm\_config\_t \* *config* )

The default values are:

```
* config->enableDebugMode = false;
* config->enableWait = false;
* config->reloadSelect = kPWM_LocalReload;
* config->faultFilterCount = 0;
* config->faultFilterPeriod = 0;
* config->clockSource = kPWM_BusClock;
* config->prescale = kPWM_Prescale_Divide_1;
* config->initializationControl = kPWM_Initialize_LocalSync;
* config->forceTrigger = kPWM_Force_Local;
* config->reloadFrequency = kPWM_LoadEveryOpportunity;
* config->reloadLogic = kPWM_ReloadImmediate;
* config->pairOperation = kPWM_Independent;
*
```

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>config</i> | Pointer to user's PWM config structure. |
|---------------|-----------------------------------------|

### 34.6.4 status\_t PWM\_SetupPwm ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, const pwm\_signal\_param\_t \* *chnlParams*, uint8\_t *numOfChnls*, pwm\_mode\_t *mode*, uint32\_t *pwmFreq\_Hz*, uint32\_t *srcClock\_Hz* )

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                    |                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | PWM peripheral base address                                                                                                                                      |
| <i>subModule</i>   | PWM submodule to configure                                                                                                                                       |
| <i>chnlParams</i>  | Array of PWM channel parameters to configure the channel(s)                                                                                                      |
| <i>numOfChnls</i>  | Number of channels to configure, this should be the size of the array passed in. Array size should not be more than 2 as each submodule has 2 pins to output PWM |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">pwm_mode_t</a>                                                                                  |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                                                                                                       |
| <i>srcClock_Hz</i> | PWM main counter clock in Hz.                                                                                                                                    |

## Returns

Returns `kStatusFail` if there was error setting up the signal; `kStatusSuccess` otherwise

**34.6.5 void PWM\_UpdatePwmDutycycle ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmSignal*, pwm\_mode\_t *currPwmMode*, uint8\_t *dutyCyclePercent* )**

The function updates the PWM dutycycle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                          |                                                                                                                               |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                                   |
| <i>subModule</i>         | PWM submodule to configure                                                                                                    |
| <i>pwmSignal</i>         | Signal (PWM A or PWM B) to update                                                                                             |
| <i>currPwmMode</i>       | The current PWM mode set during PWM setup                                                                                     |
| <i>dutyCycle-Percent</i> | New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

**34.6.6 void PWM\_SetupInputCapture ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, const pwm\_input\_capture\_param\_t \* *inputCaptureParams* )**

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

## Function Documentation

Parameters

|                            |                                              |
|----------------------------|----------------------------------------------|
| <i>base</i>                | PWM peripheral base address                  |
| <i>subModule</i>           | PWM submodule to configure                   |
| <i>pwmChannel</i>          | Channel in the submodule to setup            |
| <i>inputCapture-Params</i> | Parameters passed in to set up the input pin |

**34.6.7 void PWM\_SetupFaults ( PWM\_Type \* *base*, pwm\_fault\_input\_t *faultNum*, const pwm\_fault\_param\_t \* *faultParams* )**

PWM has 4 fault inputs.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | PWM peripheral base address               |
| <i>faultNum</i>    | PWM fault to configure.                   |
| <i>faultParams</i> | Pointer to the PWM fault config structure |

**34.6.8 void PWM\_SetupForceSignal ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, pwm\_force\_signal\_t *mode* )**

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>base</i>       | PWM peripheral base address                    |
| <i>subModule</i>  | PWM submodule to configure                     |
| <i>pwmChannel</i> | Channel to configure                           |
| <i>mode</i>       | Signal to output when a FORCE_OUT is triggered |

**34.6.9 void PWM\_EnableInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint32\_t *mask* )**

## Parameters

|                  |                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                         |
| <i>subModule</i> | PWM submodule to configure                                                                                          |
| <i>mask</i>      | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a> |

### 34.6.10 void PWM\_DisableInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint32\_t *mask* )

## Parameters

|                  |                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                         |
| <i>subModule</i> | PWM submodule to configure                                                                                          |
| <i>mask</i>      | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a> |

### 34.6.11 uint32\_t PWM\_GetEnabledInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule* )

## Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

## Returns

The enabled interrupts. This is the logical OR of members of the enumeration [pwm\\_interrupt\\_enable\\_t](#)

### 34.6.12 uint32\_t PWM\_GetStatusFlags ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule* )

## Function Documentation

### Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

### Returns

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

### 34.6.13 void PWM\_ClearStatusFlags ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint32\_t *mask* )

### Parameters

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

### 34.6.14 static void PWM\_StartTimer ( PWM\_Type \* *base*, uint8\_t *subModulesToStart* ) [inline], [static]

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

### Parameters

|                          |                                                                                                                  |
|--------------------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                      |
| <i>subModulesToStart</i> | PWM submodules to start. This is a logical OR of members of the enumeration <a href="#">pwm_module_control_t</a> |

### 34.6.15 static void PWM\_StopTimer ( PWM\_Type \* *base*, uint8\_t *subModulesToStop* ) [inline], [static]

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.



## Parameters

|                          |                                                                                                                  |
|--------------------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                      |
| <i>subModulesTo-Stop</i> | PWM submodules to stop. This is a logical OR of members of the enumeration <a href="#">pwm-_module_control_t</a> |

**34.6.16 static void PWM\_OutputTriggerEnable ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_value\_register\_t *valueRegister*, bool *activate* ) [inline], [static]**

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

## Parameters

|                      |                                                      |
|----------------------|------------------------------------------------------|
| <i>base</i>          | PWM peripheral base address                          |
| <i>subModule</i>     | PWM submodule to configure                           |
| <i>valueRegister</i> | Value register that will activate the trigger        |
| <i>activate</i>      | true: Enable the trigger; false: Disable the trigger |

**34.6.17 static void PWM\_SetupSwCtrlOut ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, bool *value* ) [inline], [static]**

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

## Parameters

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>base</i>       | PWM peripheral base address                      |
| <i>subModule</i>  | PWM submodule to configure                       |
| <i>pwmChannel</i> | Channel to configure                             |
| <i>value</i>      | true: Supply a logic 1, false: Supply a logic 0. |

## Function Documentation

### 34.6.18 `static void PWM_SetPwmLdok ( PWM_Type * base, uint8_t subModulesToUpdate, bool value ) [inline], [static]`

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if kPWM\_ReloadImmediate option was chosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

Parameters

|                           |                                                                                                                                        |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                                                                            |
| <i>subModulesToUpdate</i> | PWM submodules to update with buffered values. This is a logical OR of members of the enumeration <a href="#">pwm_module_control_t</a> |
| <i>value</i>              | true: Set LDOK bit for the submodule list; false: Clear LDOK bit                                                                       |

# Chapter 35

## PXP: Pixel Pipeline

### 35.1 Overview

The MCUXpresso SDK provides a driver for the Pixel Pipeline (PXP)

The PXP is used to process graphics buffers or composite video and graphics data before sending to an LCD display or TV encoder. The PXP driver only provides functional APIs. It does not maintain software level state, so that the APIs could be involved directly to any upper layer graphics framework easily.

To use the PXP driver, call [PXP\\_Init](#) first to enable and initialize the peripheral. Generally, call the PXP driver APIs to configure input buffer, output buffer, and other settings such as flip, rotate, then call [PXP\\_Start](#), thus the PXP starts the processing. When finished, the flag [kPXP\\_CompleteFlag](#) asserts. PXP also supports operation queuing, it means that a new operation could be submitted to PXP while the current PXP operation is running. When current operation finished, the new operation configurations are loaded to PXP register and new processing starts.

### 35.2 Typical use case

#### 35.2.1 PXP normal operation

This example shows how to perform vertical flip to process surface and save to output buffer. The input and output buffer pixel format are RGB888.

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

#### 35.2.2 PXP operation queue

This example shows how to perform vertical flip to process surface using operation queue. The input and output buffer pixel format are RGB888.

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

### Data Structures

- struct [pxp\\_output\\_buffer\\_config\\_t](#)  
*PXP output buffer configuration. [More...](#)*
- struct [pxp\\_ps\\_buffer\\_config\\_t](#)  
*PXP process surface buffer configuration. [More...](#)*
- struct [pxp\\_as\\_buffer\\_config\\_t](#)  
*PXP alpha surface buffer configuration. [More...](#)*
- struct [pxp\\_as\\_blend\\_config\\_t](#)  
*PXP alpha surface blending configuration. [More...](#)*
- struct [pxp\\_csc2\\_config\\_t](#)

## Typical use case

- *PXP CSC2 configuration. [More...](#)*  
struct [pxp\\_dither\\_final\\_lut\\_data\\_t](#)  
*PXP dither final LUT data. [More...](#)*
- struct [pxp\\_dither\\_config\\_t](#)  
*PXP dither configuration. [More...](#)*

## Enumerations

- enum [\\_pxp\\_interrupt\\_enable](#) {  
[kPXP\\_CommandLoadInterruptEnable](#) = PXP\_CTRL\_NEXT\_IRQ\_ENABLE\_MASK,  
[kPXP\\_CompleteInterruptEnable](#) = PXP\_CTRL\_IRQ\_ENABLE\_MASK }  
*PXP interrupts to enable.*
- enum [\\_pxp\\_flags](#) {  
[kPXP\\_CommandLoadFlag](#) = PXP\_STAT\_NEXT\_IRQ\_MASK,  
[kPXP\\_CompleteFlag](#) = PXP\_STAT\_IRQ0\_MASK,  
[kPXP\\_Axi0ReadErrorFlag](#) = PXP\_STAT\_AXI\_READ\_ERROR\_0\_MASK,  
[kPXP\\_Axi0WriteErrorFlag](#) = PXP\_STAT\_AXI\_WRITE\_ERROR\_0\_MASK }  
*PXP status flags.*
- enum [pxp\\_flip\\_mode\\_t](#) {  
[kPXP\\_FlipDisable](#) = 0U,  
[kPXP\\_FlipHorizontal](#) = 0x01U,  
[kPXP\\_FlipVertical](#) = 0x02U,  
[kPXP\\_FlipBoth](#) = 0x03U }  
*PXP output flip mode.*
- enum [pxp\\_rotate\\_position\\_t](#) {  
[kPXP\\_RotateOutputBuffer](#) = 0U,  
[kPXP\\_RotateProcessSurface](#) }  
*PXP rotate mode.*
- enum [pxp\\_rotate\\_degree\\_t](#) {  
[kPXP\\_Rotate0](#) = 0U,  
[kPXP\\_Rotate90](#),  
[kPXP\\_Rotate180](#),  
[kPXP\\_Rotate270](#) }  
*PXP rotate degree.*
- enum [pxp\\_interlaced\\_output\\_mode\\_t](#) {  
[kPXP\\_OutputProgressive](#) = 0U,  
[kPXP\\_OutputField0](#),  
[kPXP\\_OutputField1](#),  
[kPXP\\_OutputInterlaced](#) }  
*PXP interlaced output mode.*
- enum [pxp\\_output\\_pixel\\_format\\_t](#) {

```

kPXP_OutputPixelFormatARGB8888 = 0x0,
kPXP_OutputPixelFormatRGB888 = 0x4,
kPXP_OutputPixelFormatRGB888P = 0x5,
kPXP_OutputPixelFormatARGB1555 = 0x8,
kPXP_OutputPixelFormatARGB4444 = 0x9,
kPXP_OutputPixelFormatRGB555 = 0xC,
kPXP_OutputPixelFormatRGB444 = 0xD,
kPXP_OutputPixelFormatRGB565 = 0xE,
kPXP_OutputPixelFormatYUV1P444 = 0x10,
kPXP_OutputPixelFormatUYVY1P422 = 0x12,
kPXP_OutputPixelFormatVYUY1P422 = 0x13,
kPXP_OutputPixelFormatY8 = 0x14,
kPXP_OutputPixelFormatY4 = 0x15,
kPXP_OutputPixelFormatYUV2P422 = 0x18,
kPXP_OutputPixelFormatYUV2P420 = 0x19,
kPXP_OutputPixelFormatYVU2P422 = 0x1A,
kPXP_OutputPixelFormatYVU2P420 = 0x1B }

```

*PXP output buffer format.*

- enum `pxp_ps_pixel_format_t` {
 

```

kPXP_PsPixelFormatRGB888 = 0x4,
kPXP_PsPixelFormatRGB555 = 0xC,
kPXP_PsPixelFormatRGB444 = 0xD,
kPXP_PsPixelFormatRGB565 = 0xE,
kPXP_PsPixelFormatYUV1P444 = 0x10,
kPXP_PsPixelFormatUYVY1P422 = 0x12,
kPXP_PsPixelFormatVYUY1P422 = 0x13,
kPXP_PsPixelFormatY8 = 0x14,
kPXP_PsPixelFormatY4 = 0x15,
kPXP_PsPixelFormatYUV2P422 = 0x18,
kPXP_PsPixelFormatYUV2P420 = 0x19,
kPXP_PsPixelFormatYVU2P422 = 0x1A,
kPXP_PsPixelFormatYVU2P420 = 0x1B,
kPXP_PsPixelFormatYVU422 = 0x1E,
kPXP_PsPixelFormatYVU420 = 0x1F }

```

*PXP process surface buffer pixel format.*

- enum `pxp_as_pixel_format_t` {
 

```

kPXP_AsPixelFormatARGB8888 = 0x0,
kPXP_AsPixelFormatRGB888 = 0x4,
kPXP_AsPixelFormatARGB1555 = 0x8,
kPXP_AsPixelFormatARGB4444 = 0x9,
kPXP_AsPixelFormatRGB555 = 0xC,
kPXP_AsPixelFormatRGB444 = 0xD,
kPXP_AsPixelFormatRGB565 = 0xE }

```

*PXP alpha surface buffer pixel format.*

- enum `pxp_alpha_mode_t` {

## Typical use case

```
kPXP_AlphaEmbedded,  
kPXP_AlphaOverride,  
kPXP_AlphaMultiply,  
kPXP_AlphaRop }
```

*PXP alpha mode during blending.*

- enum `pxp_rop_mode_t` {  
kPXP\_RopMaskAs = 0x0,  
kPXP\_RopMaskNotAs = 0x1,  
kPXP\_RopMaskAsNot = 0x2,  
kPXP\_RopMergeAs = 0x3,  
kPXP\_RopMergeNotAs = 0x4,  
kPXP\_RopMergeAsNot = 0x5,  
kPXP\_RopNotCopyAs = 0x6,  
kPXP\_RopNot = 0x7,  
kPXP\_RopNotMaskAs = 0x8,  
kPXP\_RopNotMergeAs = 0x9,  
kPXP\_RopXorAs = 0xA,  
kPXP\_RopNotXorAs = 0xB }

*PXP ROP mode during blending.*

- enum `pxp_block_size_t` {  
kPXP\_BlockSize8 = 0U,  
kPXP\_BlockSize16 }

*PXP process block size.*

- enum `pxp_csc1_mode_t` {  
kPXP\_Csc1YUV2RGB = 0U,  
kPXP\_Csc1YCbCr2RGB }

*PXP CSC1 mode.*

- enum `pxp_csc2_mode_t` {  
kPXP\_Csc2YUV2RGB = 0U,  
kPXP\_Csc2YCbCr2RGB,  
kPXP\_Csc2RGB2YUV,  
kPXP\_Csc2RGB2YCbCr }

*PXP CSC2 mode.*

- enum `pxp_ram_t` {  
kPXP\_RamDither0Lut = 0U,  
kPXP\_RamDither1Lut = 3U,  
kPXP\_RamDither2Lut = 4U }

*PXP internal memory.*

- enum `_pxp_dither_mode` {  
kPXP\_DitherPassThrough = 0U,  
kPXP\_DitherOrdered = 3U,  
kPXP\_DitherQuantOnly = 4U }

*PXP dither mode.*

- enum `_pxp_dither_lut_mode` {  
kPXP\_DitherLutOff = 0U,  
kPXP\_DitherLutPreDither,

- `kPXP_DitherLutPostDither` }  
*PXP dither LUT mode.*
- enum `_pxp_dither_matrix_size` {  
`kPXP_DitherMatrix8` = 1,  
`kPXP_DitherMatrix16` }  
*PXP dither matrix size.*

## Driver version

- #define `FSL_PXP_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void `PXP_Init` (PXP\_Type \*base)  
*Initialize the PXP.*
- void `PXP_Deinit` (PXP\_Type \*base)  
*De-initialize the PXP.*
- void `PXP_Reset` (PXP\_Type \*base)  
*Reset the PXP.*

## Global operations

- static void `PXP_Start` (PXP\_Type \*base)  
*Start process.*
- static void `PXP_EnableLcdHandShake` (PXP\_Type \*base, bool enable)  
*Enable or disable LCD hand shake.*
- static void `PXP_EnableContinuousRun` (PXP\_Type \*base, bool enable)  
*Enable or disable continuous run.*
- static void `PXP_SetProcessBlockSize` (PXP\_Type \*base, `pxp_block_size_t` size)  
*Set the PXP processing block size.*

## Status

- static uint32\_t `PXP_GetStatusFlags` (PXP\_Type \*base)  
*Gets PXP status flags.*
- static void `PXP_ClearStatusFlags` (PXP\_Type \*base, uint32\_t statusMask)  
*Clears status flags with the provided mask.*
- static uint8\_t `PXP_GetAxiErrorId` (PXP\_Type \*base, uint8\_t axiIndex)  
*Gets the AXI ID of the failing bus operation.*

## Interrupts

- static void `PXP_EnableInterrupts` (PXP\_Type \*base, uint32\_t mask)  
*Enables PXP interrupts according to the provided mask.*
- static void `PXP_DisableInterrupts` (PXP\_Type \*base, uint32\_t mask)  
*Disables PXP interrupts according to the provided mask.*

## Typical use case

### Alpha surface

- void [PXP\\_SetAlphaSurfaceBufferConfig](#) (PXP\_Type \*base, const [pxp\\_as\\_buffer\\_config\\_t](#) \*config)  
*Set the alpha surface input buffer configuration.*
- void [PXP\\_SetAlphaSurfaceBlendConfig](#) (PXP\_Type \*base, const [pxp\\_as\\_blend\\_config\\_t](#) \*config)  
*Set the alpha surface blending configuration.*
- void [PXP\\_SetAlphaSurfaceOverlayColorKey](#) (PXP\_Type \*base, uint32\_t colorKeyLow, uint32\_t colorKeyHigh)  
*Set the alpha surface overlay color key.*
- static void [PXP\\_EnableAlphaSurfaceOverlayColorKey](#) (PXP\_Type \*base, bool enable)  
*Enable or disable the alpha surface color key.*
- void [PXP\\_SetAlphaSurfacePosition](#) (PXP\_Type \*base, uint16\_t upperLeftX, uint16\_t upperLeftY, uint16\_t lowerRightX, uint16\_t lowerRightY)  
*Set the alpha surface position in output buffer.*

### Process surface

- static void [PXP\\_SetProcessSurfaceBackgroundColor](#) (PXP\_Type \*base, uint32\_t backGroundColor)  
*Set the back ground color of PS.*
- void [PXP\\_SetProcessSurfaceBufferConfig](#) (PXP\_Type \*base, const [pxp\\_ps\\_buffer\\_config\\_t](#) \*config)  
*Set the process surface input buffer configuration.*
- void [PXP\\_SetProcessSurfaceScaler](#) (PXP\_Type \*base, uint16\_t inputWidth, uint16\_t inputHeight, uint16\_t outputWidth, uint16\_t outputHeight)  
*Set the process surface scaler configuration.*
- void [PXP\\_SetProcessSurfacePosition](#) (PXP\_Type \*base, uint16\_t upperLeftX, uint16\_t upperLeftY, uint16\_t lowerRightX, uint16\_t lowerRightY)  
*Set the process surface position in output buffer.*
- void [PXP\\_SetProcessSurfaceColorKey](#) (PXP\_Type \*base, uint32\_t colorKeyLow, uint32\_t colorKeyHigh)  
*Set the process surface color key.*

### Output buffer

- void [PXP\\_SetOutputBufferConfig](#) (PXP\_Type \*base, const [pxp\\_output\\_buffer\\_config\\_t](#) \*config)  
*Set the PXP outpt buffer configuration.*
- static void [PXP\\_SetOverwrittenAlphaValue](#) (PXP\_Type \*base, uint8\_t alpha)  
*Set the global overwritten alpha value.*
- static void [PXP\\_EnableOverWrittenAlpha](#) (PXP\_Type \*base, bool enable)  
*Enable or disable the global overwritten alpha value.*
- static void [PXP\\_SetRotateConfig](#) (PXP\_Type \*base, [pxp\\_rotate\\_position\\_t](#) position, [pxp\\_rotate\\_degree\\_t](#) degree, [pxp\\_flip\\_mode\\_t](#) flipMode)  
*Set the rotation configuration.*

### Command queue

- static void [PXP\\_SetNextCommand](#) (PXP\_Type \*base, void \*commandAddr)  
*Set the next command.*
- static bool [PXP\\_IsNextCommandPending](#) (PXP\_Type \*base)



- *Check whether the next command is pending.*
- static void [PXP\\_CancelNextCommand](#) (PXP\_Type \*base)  
*Cancel command set by [PXP\\_SetNextCommand](#).*

## Color space conversion

- void [PXP\\_SetCsc1Mode](#) (PXP\_Type \*base, [pxp\\_csc1\\_mode\\_t](#) mode)  
*Set the CSC1 mode.*
- static void [PXP\\_EnableCsc1](#) (PXP\_Type \*base, bool enable)  
*Enable or disable the CSC1.*

## 35.3 Data Structure Documentation

### 35.3.1 struct [pxp\\_output\\_buffer\\_config\\_t](#)

#### Data Fields

- [pxp\\_output\\_pixel\\_format\\_t](#) `pixelFormat`  
*Output buffer pixel format.*
- [pxp\\_interlaced\\_output\\_mode\\_t](#) `interlacedMode`  
*Interlaced output mode.*
- [uint32\\_t](#) `buffer0Addr`  
*Output buffer 0 address.*
- [uint32\\_t](#) `buffer1Addr`  
*Output buffer 1 address, used for UV data in YUV 2-plane mode, or field 1 in output interlaced mode.*
- [uint16\\_t](#) `pitchBytes`  
*Number of bytes between two vertically adjacent pixels.*
- [uint16\\_t](#) `width`  
*Pixels per line.*
- [uint16\\_t](#) `height`  
*How many lines in output buffer.*

## Data Structure Documentation

### 35.3.1.0.0.31 Field Documentation

35.3.1.0.0.31.1 `pxp_output_pixel_format_t` `pxp_output_buffer_config_t::pixelFormat`

35.3.1.0.0.31.2 `pxp_interlaced_output_mode_t` `pxp_output_buffer_config_t::interlacedMode`

35.3.1.0.0.31.3 `uint32_t` `pxp_output_buffer_config_t::buffer0Addr`

35.3.1.0.0.31.4 `uint32_t` `pxp_output_buffer_config_t::buffer1Addr`

35.3.1.0.0.31.5 `uint16_t` `pxp_output_buffer_config_t::pitchBytes`

35.3.1.0.0.31.6 `uint16_t` `pxp_output_buffer_config_t::width`

35.3.1.0.0.31.7 `uint16_t` `pxp_output_buffer_config_t::height`

### 35.3.2 struct `pxp_ps_buffer_config_t`

#### Data Fields

- `pxp_ps_pixel_format_t` `pixelFormat`  
*PS buffer pixel format.*
- `bool` `swapByte`  
*For each 16 bit word, set true to swap the two bytes.*
- `uint32_t` `bufferAddr`  
*Input buffer address for the first panel.*
- `uint32_t` `bufferAddrU`  
*Input buffer address for the second panel.*
- `uint32_t` `bufferAddrV`  
*Input buffer address for the third panel.*
- `uint16_t` `pitchBytes`  
*Number of bytes between two vertically adjacent pixels.*

**35.3.2.0.0.32 Field Documentation****35.3.2.0.0.32.1** `pxp_ps_pixel_format_t` `pxp_ps_buffer_config_t::pixelFormat`**35.3.2.0.0.32.2** `bool` `pxp_ps_buffer_config_t::swapByte`**35.3.2.0.0.32.3** `uint32_t` `pxp_ps_buffer_config_t::bufferAddr`**35.3.2.0.0.32.4** `uint32_t` `pxp_ps_buffer_config_t::bufferAddrU`**35.3.2.0.0.32.5** `uint32_t` `pxp_ps_buffer_config_t::bufferAddrV`**35.3.2.0.0.32.6** `uint16_t` `pxp_ps_buffer_config_t::pitchBytes`**35.3.3 struct `pxp_as_buffer_config_t`****Data Fields**

- `pxp_as_pixel_format_t` `pixelFormat`  
*AS buffer pixel format.*
- `uint32_t` `bufferAddr`  
*Input buffer address.*
- `uint16_t` `pitchBytes`  
*Number of bytes between two vertically adjacent pixels.*

**35.3.3.0.0.33 Field Documentation****35.3.3.0.0.33.1** `pxp_as_pixel_format_t` `pxp_as_buffer_config_t::pixelFormat`**35.3.3.0.0.33.2** `uint32_t` `pxp_as_buffer_config_t::bufferAddr`**35.3.3.0.0.33.3** `uint16_t` `pxp_as_buffer_config_t::pitchBytes`**35.3.4 struct `pxp_as_blend_config_t`****Data Fields**

- `uint8_t` `alpha`  
*User defined alpha value, only used when `alphaMode` is `kPXP_AlphaOverride` or `kPXP_AlphaRop`.*
- `bool` `invertAlpha`  
*Set true to invert the alpha.*
- `pxp_alpha_mode_t` `alphaMode`  
*Alpha mode.*
- `pxp_rop_mode_t` `ropMode`  
*ROP mode, only valid when `alphaMode` is `kPXP_AlphaRop`.*

## Data Structure Documentation

### 35.3.4.0.0.34 Field Documentation

35.3.4.0.0.34.1 `uint8_t pxp_as_blend_config_t::alpha`

35.3.4.0.0.34.2 `bool pxp_as_blend_config_t::invertAlpha`

35.3.4.0.0.34.3 `pxp_alpha_mode_t pxp_as_blend_config_t::alphaMode`

35.3.4.0.0.34.4 `pxp_rop_mode_t pxp_as_blend_config_t::ropMode`

### 35.3.5 `struct pxp_csc2_config_t`

Converting from YUV/YCbCr color spaces to the RGB color space uses the following equation structure:

$$R = A1(Y+D1) + A2(U+D2) + A3(V+D3) \quad G = B1(Y+D1) + B2(U+D2) + B3(V+D3) \quad B = C1(Y+D1) + C2(U+D2) + C3(V+D3)$$

Converting from the RGB color space to YUV/YCbCr color spaces uses the following equation structure:

$$Y = A1*R + A2*G + A3*B + D1 \quad U = B1*R + B2*G + B3*B + D2 \quad V = C1*R + C2*G + C3*B + D3$$

## Data Fields

- `pxp_csc2_mode_t mode`  
*Conversion mode.*
- float `A1`  
*A1.*
- float `A2`  
*A2.*
- float `A3`  
*A3.*
- float `B1`  
*B1.*
- float `B2`  
*B2.*
- float `B3`  
*B3.*
- float `C1`  
*C1.*
- float `C2`  
*C2.*
- float `C3`  
*C3.*
- `int16_t D1`  
*D1.*
- `int16_t D2`  
*D2.*
- `int16_t D3`  
*D3.*

**35.3.5.0.0.35 Field Documentation****35.3.5.0.0.35.1** `pxp_csc2_mode_t` `pxp_csc2_config_t::mode`**35.3.5.0.0.35.2** `float` `pxp_csc2_config_t::A1`**35.3.5.0.0.35.3** `float` `pxp_csc2_config_t::A2`**35.3.5.0.0.35.4** `float` `pxp_csc2_config_t::A3`**35.3.5.0.0.35.5** `float` `pxp_csc2_config_t::B1`**35.3.5.0.0.35.6** `float` `pxp_csc2_config_t::B2`**35.3.5.0.0.35.7** `float` `pxp_csc2_config_t::B3`**35.3.5.0.0.35.8** `float` `pxp_csc2_config_t::C1`**35.3.5.0.0.35.9** `float` `pxp_csc2_config_t::C2`**35.3.5.0.0.35.10** `float` `pxp_csc2_config_t::C3`**35.3.5.0.0.35.11** `int16_t` `pxp_csc2_config_t::D1`**35.3.5.0.0.35.12** `int16_t` `pxp_csc2_config_t::D2`**35.3.5.0.0.35.13** `int16_t` `pxp_csc2_config_t::D3`**35.3.6 struct `pxp_dither_final_lut_data_t`****Data Fields**

- `uint32_t` [data\\_3\\_0](#)  
*Data 3 to data 0.*
- `uint32_t` [data\\_7\\_4](#)  
*Data 7 to data 4.*
- `uint32_t` [data\\_11\\_8](#)  
*Data 11 to data 8.*
- `uint32_t` [data\\_15\\_12](#)  
*Data 15 to data 12.*

**35.3.6.0.0.36 Field Documentation****35.3.6.0.0.36.1** `uint32_t` `pxp_dither_final_lut_data_t::data_3_0`

Data 0 is the least significant byte.

**35.3.6.0.0.36.2** `uint32_t` `pxp_dither_final_lut_data_t::data_7_4`

Data 4 is the least significant byte.

## Data Structure Documentation

### 35.3.6.0.0.36.3 uint32\_t pxp\_dither\_final\_lut\_data\_t::data\_11\_8

Data 8 is the least significant byte.

### 35.3.6.0.0.36.4 uint32\_t pxp\_dither\_final\_lut\_data\_t::data\_15\_12

Data 12 is the least significant byte.

## 35.3.7 struct pxp\_dither\_config\_t

### Data Fields

- uint32\_t `enableDither0`: 1  
*Enable dither engine 0 or not, set 1 to enable, 0 to disable.*
- uint32\_t `enableDither1`: 1  
*Enable dither engine 1 or not, set 1 to enable, 0 to disable.*
- uint32\_t `enableDither2`: 1  
*Enable dither engine 2 or not, set 1 to enable, 0 to disable.*
- uint32\_t `ditherMode0`: 3  
*Dither mode for dither engine 0.*
- uint32\_t `ditherMode1`: 3  
*Dither mode for dither engine 1.*
- uint32\_t `ditherMode2`: 3  
*Dither mode for dither engine 2.*
- uint32\_t `quantBitNum`: 3  
*Number of bits quantize down to, the valid value is 1~7.*
- uint32\_t `lutMode`: 2  
*How to use the memory LUT, see [\\_pxp\\_dither\\_lut\\_mode](#).*
- uint32\_t `idxMatrixSize0`: 2  
*Size of index matrix used for dither for dither engine 0, see [\\_pxp\\_dither\\_matrix\\_size](#).*
- uint32\_t `idxMatrixSize1`: 2  
*Size of index matrix used for dither for dither engine 1, see [\\_pxp\\_dither\\_matrix\\_size](#).*
- uint32\_t `idxMatrixSize2`: 2  
*Size of index matrix used for dither for dither engine 2, see [\\_pxp\\_dither\\_matrix\\_size](#).*
- uint32\_t `enableFinalLut`: 1  
*Enable the final LUT, set 1 to enable, 0 to disable.*

### 35.3.7.0.0.37 Field Documentation

#### 35.3.7.0.0.37.1 uint32\_t pxp\_dither\_config\_t::enableDither0

#### 35.3.7.0.0.37.2 uint32\_t pxp\_dither\_config\_t::enableDither1

#### 35.3.7.0.0.37.3 uint32\_t pxp\_dither\_config\_t::enableDither2

#### 35.3.7.0.0.37.4 uint32\_t pxp\_dither\_config\_t::ditherMode0

See [\\_pxp\\_dither\\_mode](#).

**35.3.7.0.0.37.5** `uint32_t pxp_dither_config_t::ditherMode1`

See [\\_pxp\\_dither\\_mode](#).

**35.3.7.0.0.37.6** `uint32_t pxp_dither_config_t::ditherMode2`

See [\\_pxp\\_dither\\_mode](#).

**35.3.7.0.0.37.7** `uint32_t pxp_dither_config_t::quantBitNum`

**35.3.7.0.0.37.8** `uint32_t pxp_dither_config_t::lutMode`

This must be set to `kPXP_DitherLutOff` if any dither engine uses `kPXP_DitherOrdered` mode.

**35.3.7.0.0.37.9** `uint32_t pxp_dither_config_t::idxMatrixSize0`

**35.3.7.0.0.37.10** `uint32_t pxp_dither_config_t::idxMatrixSize1`

**35.3.7.0.0.37.11** `uint32_t pxp_dither_config_t::idxMatrixSize2`

**35.3.7.0.0.37.12** `uint32_t pxp_dither_config_t::enableFinalLut`

## 35.4 Enumeration Type Documentation

### 35.4.1 `enum _pxp_interrupt_enable`

Enumerator

***kPXP\_CommandLoadInterruptEnable*** Interrupt to show that the command set by `PXP_SetNextCommand` has been loaded.

***kPXP\_CompleteInterruptEnable*** PXP process completed.

### 35.4.2 `enum _pxp_flags`

Note

These enumerations are meant to be OR'd together to form a bit mask.

Enumerator

***kPXP\_CommandLoadFlag*** The command set by `PXP_SetNextCommand` has been loaded, could set new command.

***kPXP\_CompleteFlag*** PXP process completed.

***kPXP\_Axi0ReadErrorFlag*** PXP encountered an AXI read error and processing has been terminated.

***kPXP\_Axi0WriteErrorFlag*** PXP encountered an AXI write error and processing has been terminated.

## Enumeration Type Documentation

### 35.4.3 enum pxp\_flip\_mode\_t

Enumerator

- kPXP\_FlipDisable* Flip disable.
- kPXP\_FlipHorizontal* Horizontal flip.
- kPXP\_FlipVertical* Vertical flip.
- kPXP\_FlipBoth* Flip both directions.

### 35.4.4 enum pxp\_rotate\_position\_t

Enumerator

- kPXP\_RotateOutputBuffer* Rotate the output buffer.
- kPXP\_RotateProcessSurface* Rotate the process surface.

### 35.4.5 enum pxp\_rotate\_degree\_t

Enumerator

- kPXP\_Rotate0* Clock wise rotate 0 deg.
- kPXP\_Rotate90* Clock wise rotate 90 deg.
- kPXP\_Rotate180* Clock wise rotate 180 deg.
- kPXP\_Rotate270* Clock wise rotate 270 deg.

### 35.4.6 enum pxp\_interlaced\_output\_mode\_t

Enumerator

- kPXP\_OutputProgressive* All data written in progressive format to output buffer 0.
- kPXP\_OutputField0* Only write field 0 data to output buffer 0.
- kPXP\_OutputField1* Only write field 1 data to output buffer 0.
- kPXP\_OutputInterlaced* Field 0 write to buffer 0, field 1 write to buffer 1.

### 35.4.7 enum pxp\_output\_pixel\_format\_t

Enumerator

- kPXP\_OutputPixelFormatARGB8888* 32-bit pixels with alpha.
- kPXP\_OutputPixelFormatRGB888* 32-bit pixels without alpha (unpacked 24-bit format)



*kPXP\_OutputPixelFormatRGB888P* 24-bit pixels without alpha (packed 24-bit format)  
*kPXP\_OutputPixelFormatARGB1555* 16-bit pixels with alpha.  
*kPXP\_OutputPixelFormatARGB4444* 16-bit pixels with alpha.  
*kPXP\_OutputPixelFormatRGB555* 16-bit pixels without alpha.  
*kPXP\_OutputPixelFormatRGB444* 16-bit pixels without alpha.  
*kPXP\_OutputPixelFormatRGB565* 16-bit pixels without alpha.  
*kPXP\_OutputPixelFormatYUVIP444* 32-bit pixels (1-plane XYUV unpacked).  
*kPXP\_OutputPixelFormatUYVYIP422* 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)  
*kPXP\_OutputPixelFormatVYUYIP422* 16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)  
*kPXP\_OutputPixelFormatY8* 8-bit monochrome pixels (1-plane Y luma output)  
*kPXP\_OutputPixelFormatY4* 4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)  
*kPXP\_OutputPixelFormatYUV2P422* 16-bit pixels (2-plane UV interleaved bytes)  
*kPXP\_OutputPixelFormatYUV2P420* 16-bit pixels (2-plane UV)  
*kPXP\_OutputPixelFormatYVU2P422* 16-bit pixels (2-plane VU interleaved bytes)  
*kPXP\_OutputPixelFormatYVU2P420* 16-bit pixels (2-plane VU)

### 35.4.8 enum pxp\_ps\_pixel\_format\_t

Enumerator

*kPXP\_PsPixelFormatRGB888* 32-bit pixels without alpha (unpacked 24-bit format)  
*kPXP\_PsPixelFormatRGB555* 16-bit pixels without alpha.  
*kPXP\_PsPixelFormatRGB444* 16-bit pixels without alpha.  
*kPXP\_PsPixelFormatRGB565* 16-bit pixels without alpha.  
*kPXP\_PsPixelFormatYUVIP444* 32-bit pixels (1-plane XYUV unpacked).  
*kPXP\_PsPixelFormatUYVYIP422* 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)  
*kPXP\_PsPixelFormatVYUYIP422* 16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)  
*kPXP\_PsPixelFormatY8* 8-bit monochrome pixels (1-plane Y luma output)  
*kPXP\_PsPixelFormatY4* 4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)  
*kPXP\_PsPixelFormatYUV2P422* 16-bit pixels (2-plane UV interleaved bytes)  
*kPXP\_PsPixelFormatYUV2P420* 16-bit pixels (2-plane UV)  
*kPXP\_PsPixelFormatYVU2P422* 16-bit pixels (2-plane VU interleaved bytes)  
*kPXP\_PsPixelFormatYVU2P420* 16-bit pixels (2-plane VU)  
*kPXP\_PsPixelFormatYVU422* 16-bit pixels (3-plane)  
*kPXP\_PsPixelFormatYVU420* 16-bit pixels (3-plane)

### 35.4.9 enum pxp\_as\_pixel\_format\_t

Enumerator

*kPXP\_AsPixelFormatARGB8888* 32-bit pixels with alpha.  
*kPXP\_AsPixelFormatRGB888* 32-bit pixels without alpha (unpacked 24-bit format)

## Enumeration Type Documentation

*kPXP\_AsPixelFormatARGB1555* 16-bit pixels with alpha.  
*kPXP\_AsPixelFormatARGB4444* 16-bit pixels with alpha.  
*kPXP\_AsPixelFormatRGB555* 16-bit pixels without alpha.  
*kPXP\_AsPixelFormatRGB444* 16-bit pixels without alpha.  
*kPXP\_AsPixelFormatRGB565* 16-bit pixels without alpha.

### 35.4.10 enum pxp\_alpha\_mode\_t

Enumerator

*kPXP\_AlphaEmbedded* The alpha surface pixel alpha value will be used for blend.  
*kPXP\_AlphaOverride* The user defined alpha value will be used for blend directly.  
*kPXP\_AlphaMultiply* The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set set to 200, user defined alpha set to 100, then the result alpha is  $200 * 100 / 255$ .  
*kPXP\_AlphaRop* Raster operation.

### 35.4.11 enum pxp\_rop\_mode\_t

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Enumerator

*kPXP\_RopMaskAs* AS AND PS.  
*kPXP\_RopMaskNotAs* nAS AND PS.  
*kPXP\_RopMaskAsNot* AS AND nPS.  
*kPXP\_RopMergeAs* AS OR PS.  
*kPXP\_RopMergeNotAs* nAS OR PS.  
*kPXP\_RopMergeAsNot* AS OR nPS.  
*kPXP\_RopNotCopyAs* nAS.  
*kPXP\_RopNot* nPS.  
*kPXP\_RopNotMaskAs* AS NAND PS.  
*kPXP\_RopNotMergeAs* AS NOR PS.  
*kPXP\_RopXorAs* AS XOR PS.  
*kPXP\_RopNotXorAs* AS XNOR PS.

### 35.4.12 enum pxp\_block\_size\_t

Enumerator

- kPXP\_BlockSize8* Process 8x8 pixel blocks.
- kPXP\_BlockSize16* Process 16x16 pixel blocks.

### 35.4.13 enum pxp\_csc1\_mode\_t

Enumerator

- kPXP\_Csc1YUV2RGB* YUV to RGB.
- kPXP\_Csc1YCbCr2RGB* YCbCr to RGB.

### 35.4.14 enum pxp\_csc2\_mode\_t

Enumerator

- kPXP\_Csc2YUV2RGB* YUV to RGB.
- kPXP\_Csc2YCbCr2RGB* YCbCr to RGB.
- kPXP\_Csc2RGB2YUV* RGB to YUV.
- kPXP\_Csc2RGB2YCbCr* RGB to YCbCr.

### 35.4.15 enum pxp\_ram\_t

Enumerator

- kPXP\_RamDither0Lut* Dither 0 LUT memory.
- kPXP\_RamDither1Lut* Dither 1 LUT memory.
- kPXP\_RamDither2Lut* Dither 2 LUT memory.

### 35.4.16 enum \_pxp\_dither\_mode

Enumerator

- kPXP\_DitherPassThrough* Pass through, no dither.
- kPXP\_DitherOrdered* Ordered dither.
- kPXP\_DitherQuantOnly* No dithering, only quantization.

## Function Documentation

### 35.4.17 enum \_pxp\_dither\_lut\_mode

Enumerator

*kPXP\_DitherLutOff* The LUT memory is not used for LUT, could be used as ordered dither index matrix.

*kPXP\_DitherLutPreDither* Use LUT at the pre-dither stage, The pre-dither LUT could only be used in Floyd mode or Atkinson mode, which are not supported by current PXP module.

*kPXP\_DitherLutPostDither* Use LUT at the post-dither stage.

### 35.4.18 enum \_pxp\_dither\_matrix\_size

Enumerator

*kPXP\_DitherMatrix8* The dither index matrix is 8x8.

*kPXP\_DitherMatrix16* The dither index matrix is 16x16.

## 35.5 Function Documentation

### 35.5.1 void PXP\_Init ( PXP\_Type \* *base* )

This function enables the PXP peripheral clock, and resets the PXP registers to default status.

Parameters

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

### 35.5.2 void PXP\_Deinit ( PXP\_Type \* *base* )

This function disables the PXP peripheral clock.

Parameters

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

### 35.5.3 void PXP\_Reset ( PXP\_Type \* *base* )

This function resets the PXP peripheral registers to default status.

Parameters

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

#### 35.5.4 static void PXP\_Start ( PXP\_Type \* *base* ) [*inline*], [*static*]

Start PXP process using current configuration.

Parameters

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

#### 35.5.5 static void PXP\_EnableLcdHandShake ( PXP\_Type \* *base*, bool *enable* ) [*inline*], [*static*]

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

#### 35.5.6 static void PXP\_EnableContinuousRun ( PXP\_Type \* *base*, bool *enable* ) [*inline*], [*static*]

If continuous run not enabled, [PXP\\_Start](#) starts the PXP process. When completed, PXP enters idle mode and flag [kPXP\\_CompleteFlag](#) asserts.

If continuous run enabled, the PXP will repeat based on the current configuration register settings.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

#### 35.5.7 static void PXP\_SetProcessBlockSize ( PXP\_Type \* *base*, pxp\_block\_size\_t *size* ) [*inline*], [*static*]

This function chooses the pixel block size that PXP using during process. Larger block size means better performance, but be careful that when PXP is rotating, the output must be divisible by the block size

---

## Function Documentation

selected.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | PXP peripheral base address. |
| <i>size</i> | The pixel block size.        |

### 35.5.8 static uint32\_t PXP\_GetStatusFlags ( PXP\_Type \* *base* ) [inline], [static]

This function gets all PXP status flags. The flags are returned as the logical OR value of the enumerators [\\_pxp\\_flags](#). To check a specific status, compare the return value with enumerators in [\\_pxp\\_flags](#). For example, to check whether the PXP has completed process, use like this:

```
if (kPXP_CompleteFlag & PXP_GetStatusFlags(PXP))
{
    ...
}
```

Parameters

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

Returns

PXP status flags which are OR'ed by the enumerators in the [\\_pxp\\_flags](#).

### 35.5.9 static void PXP\_ClearStatusFlags ( PXP\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

This function clears PXP status flags with a provided mask.

Parameters

|                   |                                                                                        |
|-------------------|----------------------------------------------------------------------------------------|
| <i>base</i>       | PXP peripheral base address.                                                           |
| <i>statusMask</i> | The status flags to be cleared; it is logical OR value of <a href="#">_pxp_flags</a> . |

### 35.5.10 static uint8\_t PXP\_GetAxiErrorId ( PXP\_Type \* *base*, uint8\_t *axiIndex* ) [inline], [static]

## Function Documentation

### Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | PXP peripheral base address.                                                                  |
| <i>axiIndex</i> | Whitch AXI to get <ul style="list-style-type: none"><li>• 0: AXI0</li><li>• 1: AXI1</li></ul> |

### Returns

The AXI ID of the failing bus operation.

### 35.5.11 `static void PXP_EnableInterrupts ( PXP_Type * base, uint32_t mask )` `[inline], [static]`

This function enables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_pxp\\_interrupt\\_enable](#). For example, to enable PXP process complete interrupt and command loaded interrupt, do the following.

```
PXP_EnableInterrupts(PXP, kPXP_CommandLoadInterruptEnable  
| kPXP_CompleteInterruptEnable);
```

### Parameters

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>base</i> | PXP peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_pxp_interrupt_enable</a> . |

### 35.5.12 `static void PXP_DisableInterrupts ( PXP_Type * base, uint32_t mask )` `[inline], [static]`

This function disables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_pxp\\_interrupt\\_enable](#).

### Parameters

---



|             |                                                                               |
|-------------|-------------------------------------------------------------------------------|
| <i>base</i> | PXP peripheral base address.                                                  |
| <i>mask</i> | The interrupts to disable. Logical OR of <code>_pxp_interrupt_enable</code> . |

### 35.5.13 void PXP\_SetAlphaSurfaceBufferConfig ( PXP\_Type \* *base*, const *pxp\_as\_buffer\_config\_t* \* *config* )

Parameters

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

### 35.5.14 void PXP\_SetAlphaSurfaceBlendConfig ( PXP\_Type \* *base*, const *pxp\_as\_blend\_config\_t* \* *config* )

Parameters

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

### 35.5.15 void PXP\_SetAlphaSurfaceOverlayColorKey ( PXP\_Type \* *base*, uint32\_t *colorKeyLow*, uint32\_t *colorKeyHigh* )

If a pixel in the current overlay image with a color that falls in the range from the `colorKeyLow` to `colorKeyHigh` range, it will use the process surface pixel value for that location. If no PS image is present or if the PS image also matches its colorkey range, the PS background color is used.

Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | PXP peripheral base address. |
| <i>colorKeyLow</i>  | Color key low range.         |
| <i>colorKeyHigh</i> | Color key high range.        |

Note

Colorkey operations are higher priority than alpha or ROP operations

---

## Function Documentation

**35.5.16** `static void PXP_EnableAlphaSurfaceOverlayColorKey ( PXP_Type * base,  
bool enable ) [inline], [static]`

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

**35.5.17 void PXP\_SetAlphaSurfacePosition ( PXP\_Type \* *base*, uint16\_t *upperLeftX*, uint16\_t *upperLeftY*, uint16\_t *lowerRightX*, uint16\_t *lowerRightY* )**

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>upperLeftX</i>  | X of the upper left corner.  |
| <i>upperLeftY</i>  | Y of the upper left corner.  |
| <i>lowerRightX</i> | X of the lower right corner. |
| <i>lowerRightY</i> | Y of the lower right corner. |

**35.5.18 static void PXP\_SetProcessSurfaceBackgroundColor ( PXP\_Type \* *base*, uint32\_t *backgroundColor* ) [inline], [static]**

Parameters

|                        |                                      |
|------------------------|--------------------------------------|
| <i>base</i>            | PXP peripheral base address.         |
| <i>backgroundColor</i> | Pixel value of the background color. |

**35.5.19 void PXP\_SetProcessSurfaceBufferConfig ( PXP\_Type \* *base*, const pxp\_ps\_buffer\_config\_t \* *config* )**

Parameters

---

## Function Documentation

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

**35.5.20 void PXP\_SetProcessSurfaceScaler ( PXP\_Type \* *base*, uint16\_t *inputWidth*, uint16\_t *inputHeight*, uint16\_t *outputWidth*, uint16\_t *outputHeight* )**

The valid down scale fact is  $1/(2^{12}) \sim 16$ .

Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | PXP peripheral base address. |
| <i>inputWidth</i>   | Input image width.           |
| <i>inputHeight</i>  | Input image height.          |
| <i>outputWidth</i>  | Output image width.          |
| <i>outputHeight</i> | Output image height.         |

**35.5.21 void PXP\_SetProcessSurfacePosition ( PXP\_Type \* *base*, uint16\_t *upperLeftX*, uint16\_t *upperLeftY*, uint16\_t *lowerRightX*, uint16\_t *lowerRightY* )**

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>upperLeftX</i>  | X of the upper left corner.  |
| <i>upperLeftY</i>  | Y of the upper left corner.  |
| <i>lowerRightX</i> | X of the lower right corner. |
| <i>lowerRightY</i> | Y of the lower right corner. |

**35.5.22 void PXP\_SetProcessSurfaceColorKey ( PXP\_Type \* *base*, uint32\_t *colorKeyLow*, uint32\_t *colorKeyHigh* )**

If the PS image matches colorkey range, the PS background color is output. Set *colorKeyLow* to 0xF-FFFFFF and *colorKeyHigh* to 0 will disable the colorkeying.

Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | PXP peripheral base address. |
| <i>colorKeyLow</i>  | Color key low range.         |
| <i>colorKeyHigh</i> | Color key high range.        |

**35.5.23 void PXP\_SetOutputBufferConfig ( PXP\_Type \* *base*, const pxp\_output\_buffer\_config\_t \* *config* )**

Parameters

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

**35.5.24 static void PXP\_SetOverwrittenAlphaValue ( PXP\_Type \* *base*, uint8\_t *alpha* ) [inline], [static]**

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

|              |                              |
|--------------|------------------------------|
| <i>base</i>  | PXP peripheral base address. |
| <i>alpha</i> | The alpha value.             |

**35.5.25 static void PXP\_EnableOverWrittenAlpha ( PXP\_Type \* *base*, bool *enable* ) [inline], [static]**

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

## Function Documentation

**35.5.26 static void PXP\_SetRotateConfig ( PXP\_Type \* *base*, pxp\_rotate\_position\_t *position*, pxp\_rotate\_degree\_t *degree*, pxp\_flip\_mode\_t *flipMode* )**  
**[inline], [static]**

The PXP could rotate the process surface or the output buffer. There are two PXP versions:

- Version 1: Only has one rotate sub module, the output buffer and process surface share the same rotate sub module, which means the process surface and output buffer could not be rotate at the same time. When pass in [kPXP\\_RotateOutputBuffer](#), the process surface could not use the rotate, Also when pass in [kPXP\\_RotateProcessSurface](#), output buffer could not use the rotate.
- Version 2: Has two separate rotate sub modules, the output buffer and process surface could configure the rotation independently.

Upper layer could use the macro `PXP_SHARE_ROTATE` to check which version is. `PXP_SHARE_ROTATE=1` means version 1.

Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>base</i>     | PXP peripheral base address.             |
| <i>position</i> | Rotate process surface or output buffer. |
| <i>degree</i>   | Rotate degree.                           |
| <i>flipMode</i> | Flip mode.                               |

Note

This function is different depends on the macro `PXP_SHARE_ROTATE`.

**35.5.27 static void PXP\_SetNextCommand ( PXP\_Type \* *base*, void \* *commandAddr* )**  
**[inline], [static]**

The PXP supports a primitive ability to queue up one operation while the current operation is running. Workflow:

1. Prepare the PXP register values except STAT, CSCCOEFn, NEXT in the memory in the order they appear in the register map.
2. Call this function sets the new operation to PXP.
3. There are two methods to check whether the PXP has loaded the new operation. The first method is using [PXP\\_IsNextCommandPending](#). If there is new operation not loaded by the PXP, this function returns true. The second method is checking the flag [kPXP\\_CommandLoadFlag](#), if command loaded, this flag asserts. User could enable interrupt [kPXP\\_CommandLoadInterruptEnable](#) to get the loaded signal in interrupt way.
4. When command loaded by PXP, a new command could be set using this function.

```
uint32_t pxp_command1[48];
```

```

uint32_t pxp_command2[48];

// Prepare the register values.
pxp_command1[0] = ...;
pxp_command1[1] = ...;
// ...
pxp_command2[0] = ...;
pxp_command2[1] = ...;
// ...

// Make sure no new command pending.
while (PXP_IsNextCommandPending(PXP))
{
}

// Set new operation.
PXP_SetNextCommand(PXP, pxp_command1);

// Wait for new command loaded. Here could check @ref kPXP_CommandLoadFlag too.
while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command2);

```

## Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>commandAddr</i> | Address of the new command.  |

### 35.5.28 static bool PXP\_IsNextCommandPending ( PXP\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

True is pending, false is not.

### 35.5.29 static void PXP\_CancelNextCommand ( PXP\_Type \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

### 35.5.30 void PXP\_SetCsc1Mode ( PXP\_Type \* *base*, pxp\_csc1\_mode\_t *mode* )

The CSC1 module receives scaled YUV/YCbCr444 pixels from the scale engine and converts the pixels to the RGB888 color space. It could only be used by process surface.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | PXP peripheral base address. |
| <i>mode</i> | The conversion mode.         |

### 35.5.31 static void PXP\_EnableCsc1 ( PXP\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |



## Chapter 36

# RTWDOG: 32-bit Watchdog Timer

### 36.1 Overview

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

### 36.2 Typical use case

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

### Data Structures

- struct `rtwdog_work_mode_t`  
*Defines RTWDOG work mode. [More...](#)*
- struct `rtwdog_config_t`  
*Describes RTWDOG configuration structure. [More...](#)*

### Enumerations

- enum `rtwdog_clock_source_t` {  
    `kRTWDOG_ClockSource0` = 0U,  
    `kRTWDOG_ClockSource1` = 1U,  
    `kRTWDOG_ClockSource2` = 2U,  
    `kRTWDOG_ClockSource3` = 3U }  
*Describes RTWDOG clock source.*
- enum `rtwdog_clock_prescaler_t` {  
    `kRTWDOG_ClockPrescalerDivide1` = 0x0U,  
    `kRTWDOG_ClockPrescalerDivide256` = 0x1U }  
*Describes the selection of the clock prescaler.*
- enum `rtwdog_test_mode_t` {  
    `kRTWDOG_TestModeDisabled` = 0U,  
    `kRTWDOG_UserModeEnabled` = 1U,  
    `kRTWDOG_LowByteTest` = 2U,  
    `kRTWDOG_HighByteTest` = 3U }  
*Describes RTWDOG test mode.*
- enum `_rtwdog_interrupt_enable_t` { `kRTWDOG_InterruptEnable` = RTWDOG\_CS\_INT\_MASK }  
*RTWDOG interrupt configuration structure.*
- enum `_rtwdog_status_flags_t` {  
    `kRTWDOG_RunningFlag` = RTWDOG\_CS\_EN\_MASK,  
    `kRTWDOG_InterruptFlag` = RTWDOG\_CS\_FLG\_MASK }  
*RTWDOG status flags.*

## Typical use case

### Unlock sequence

- #define [WDOG\\_FIRST\\_WORD\\_OF\\_UNLOCK](#) (RTWDOG\_UPDATE\_KEY & 0xFFFFU)  
*First word of unlock sequence.*
- #define [WDOG\\_SECOND\\_WORD\\_OF\\_UNLOCK](#) ((RTWDOG\_UPDATE\_KEY >> 16U)& 0xFFFFU)  
*Second word of unlock sequence.*

### Refresh sequence

- #define [WDOG\\_FIRST\\_WORD\\_OF\\_REFRESH](#) (RTWDOG\_REFRESH\_KEY & 0xFFFFU)  
*First word of refresh sequence.*
- #define [WDOG\\_SECOND\\_WORD\\_OF\\_REFRESH](#) ((RTWDOG\_REFRESH\_KEY >> 16U)& 0xFFFFU)  
*Second word of refresh sequence.*

### Driver version

- #define [FSL\\_RTWDOG\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*RTWDOG driver version 2.0.0.*

### RTWDOG Initialization and De-initialization

- void [RTWDOG\\_GetDefaultConfig](#) (rtwdog\_config\_t \*config)  
*Initializes the RTWDOG configuration structure.*
- void [RTWDOG\\_Init](#) (RTWDOG\_Type \*base, const rtwdog\_config\_t \*config)  
*Initializes the RTWDOG module.*
- void [RTWDOG\\_Deinit](#) (RTWDOG\_Type \*base)  
*De-initializes the RTWDOG module.*

### RTWDOG functional Operation

- static void [RTWDOG\\_Enable](#) (RTWDOG\_Type \*base)  
*Enables the RTWDOG module.*
- static void [RTWDOG\\_Disable](#) (RTWDOG\_Type \*base)  
*Disables the RTWDOG module.*
- static void [RTWDOG\\_EnableInterrupts](#) (RTWDOG\_Type \*base, uint32\_t mask)  
*Enables the RTWDOG interrupt.*
- static void [RTWDOG\\_DisableInterrupts](#) (RTWDOG\_Type \*base, uint32\_t mask)  
*Disables the RTWDOG interrupt.*
- static uint32\_t [RTWDOG\\_GetStatusFlags](#) (RTWDOG\_Type \*base)  
*Gets the RTWDOG all status flags.*
- void [RTWDOG\\_ClearStatusFlags](#) (RTWDOG\_Type \*base, uint32\_t mask)  
*Clears the RTWDOG flag.*
- static void [RTWDOG\\_SetTimeoutValue](#) (RTWDOG\_Type \*base, uint16\_t timeoutCount)  
*Sets the RTWDOG timeout value.*
- static void [RTWDOG\\_SetWindowValue](#) (RTWDOG\_Type \*base, uint16\_t windowValue)  
*Sets the RTWDOG window value.*
- static void [RTWDOG\\_Unlock](#) (RTWDOG\_Type \*base)  
*Unlocks the RTWDOG register written.*
- static void [RTWDOG\\_Refresh](#) (RTWDOG\_Type \*base)

*Refreshes the RTWDOG timer.*

- static uint16\_t [RTWDOG\\_GetCounterValue](#) (RTWDOG\_Type \*base)  
*Gets the RTWDOG counter value.*

## 36.3 Data Structure Documentation

### 36.3.1 struct rtwdog\_work\_mode\_t

#### Data Fields

- bool [enableWait](#)  
*Enables or disables RTWDOG in wait mode.*
- bool [enableStop](#)  
*Enables or disables RTWDOG in stop mode.*
- bool [enableDebug](#)  
*Enables or disables RTWDOG in debug mode.*

### 36.3.2 struct rtwdog\_config\_t

#### Data Fields

- bool [enableRtwdog](#)  
*Enables or disables RTWDOG.*
- [rtwdog\\_clock\\_source\\_t](#) [clockSource](#)  
*Clock source select.*
- [rtwdog\\_clock\\_prescaler\\_t](#) [prescaler](#)  
*Clock prescaler value.*
- [rtwdog\\_work\\_mode\\_t](#) [workMode](#)  
*Configures RTWDOG work mode in debug stop and wait mode.*
- [rtwdog\\_test\\_mode\\_t](#) [testMode](#)  
*Configures RTWDOG test mode.*
- bool [enableUpdate](#)  
*Update write-once register enable.*
- bool [enableInterrupt](#)  
*Enables or disables RTWDOG interrupt.*
- bool [enableWindowMode](#)  
*Enables or disables RTWDOG window mode.*
- uint16\_t [windowValue](#)  
*Window value.*
- uint16\_t [timeoutValue](#)  
*Timeout value.*

## 36.4 Macro Definition Documentation

### 36.4.1 #define FSL\_RTWDOG\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## Enumeration Type Documentation

### 36.5 Enumeration Type Documentation

#### 36.5.1 enum rtwdog\_clock\_source\_t

Enumerator

- kRTWDOG\_ClockSource0* Clock source 0.
- kRTWDOG\_ClockSource1* Clock source 1.
- kRTWDOG\_ClockSource2* Clock source 2.
- kRTWDOG\_ClockSource3* Clock source 3.

#### 36.5.2 enum rtwdog\_clock\_prescaler\_t

Enumerator

- kRTWDOG\_ClockPrescalerDivide1* Divided by 1.
- kRTWDOG\_ClockPrescalerDivide256* Divided by 256.

#### 36.5.3 enum rtwdog\_test\_mode\_t

Enumerator

- kRTWDOG\_TestModeDisabled* Test Mode disabled.
- kRTWDOG\_UserModeEnabled* User Mode enabled.
- kRTWDOG\_LowByteTest* Test Mode enabled, only low byte is used.
- kRTWDOG\_HighByteTest* Test Mode enabled, only high byte is used.

#### 36.5.4 enum \_rtwdog\_interrupt\_enable\_t

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

Enumerator

- kRTWDOG\_InterruptEnable* Interrupt is generated before forcing a reset.

#### 36.5.5 enum \_rtwdog\_status\_flags\_t

This structure contains the RTWDOG status flags for use in the RTWDOG functions.

Enumerator

- kRTWDOG\_RunningFlag* Running flag, set when RTWDOG is enabled.
- kRTWDOG\_InterruptFlag* Interrupt flag, set when interrupt occurs.

## 36.6 Function Documentation

### 36.6.1 void RTWDOG\_GetDefaultConfig ( rtwdog\_config\_t \* config )

This function initializes the RTWDOG configuration structure to default values. The default values are:

```
* rtwdogConfig->enableRtdog = true;
* rtwdogConfig->clockSource = kRTWDOG_ClockSource1;
* rtwdogConfig->prescaler = kRTWDOG_ClockPrescalerDivide1;
* rtwdogConfig->workMode.enableWait = true;
* rtwdogConfig->workMode.enableStop = false;
* rtwdogConfig->workMode.enableDebug = false;
* rtwdogConfig->testMode = kRTWDOG_TestModeDisabled;
* rtwdogConfig->enableUpdate = true;
* rtwdogConfig->enableInterrupt = false;
* rtwdogConfig->enableWindowMode = false;
* rtwdogConfig->windowValue = 0U;
* rtwdogConfig->timeoutValue = 0xFFFFU;
*
```

#### Parameters

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

See Also

[rtwdog\\_config\\_t](#)

### 36.6.2 void RTWDOG\_Init ( RTWDOG\_Type \* base, const rtwdog\_config\_t \* config )

This function initializes the RTWDOG. To reconfigure the RTWDOG without forcing a reset first, enable-Update must be set to true in the configuration.

Example:

```
* rtwdog_config_t config;
* RTWDOG_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* config.enableUpdate = true;
* RTWDOG_Init(wdog_base, &config);
*
```

#### Parameters

## Function Documentation

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | RTWDOG peripheral base address.  |
| <i>config</i> | The configuration of the RTWDOG. |

### 36.6.3 void RTWDOG\_Deinit ( RTWDOG\_Type \* *base* )

This function shuts down the RTWDOG. Ensure that the WDOG\_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

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

### 36.6.4 static void RTWDOG\_Enable ( RTWDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to enable the RTWDOG. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

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

### 36.6.5 static void RTWDOG\_Disable ( RTWDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to disable the RTWDOG. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

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

### 36.6.6 static void RTWDOG\_EnableInterrupts ( RTWDOG\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function writes a value into the WDOG\_CS register to enable the RTWDOG interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not

been written in this WCT while the function is called.

## Function Documentation

### Parameters

|             |                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                            |
| <i>mask</i> | The interrupts to enable. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kRTWDOG_InterruptEnable</li></ul> |

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

This function writes a value into the WDOG\_CS register to disable the RTWDOG interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

### Parameters

|             |                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                              |
| <i>mask</i> | The interrupts to disabled. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kRTWDOG_InterruptEnable</li></ul> |

### 36.6.8 static uint32\_t RTWDOG\_GetStatusFlags ( RTWDOG\_Type \* *base* ) [inline], [static]

This function gets all status flags.

Example to get the running flag:

```
* uint32_t status;  
* status = RTWDOG_GetStatusFlags(wdog_base) &  
*         kRTWDOG_RunningFlag;  
*
```

### Parameters



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

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

[\\_rtwdog\\_status\\_flags\\_t](#)

- true: related status flag has been set.
- false: related status flag is not set.

**36.6.9 void RTWDOG\_ClearStatusFlags ( RTWDOG\_Type \* *base*, uint32\_t *mask* )**

This function clears the RTWDOG status flag.

Example to clear an interrupt flag:

```
* RTWDOG_ClearStatusFlags (wdog_base,
    kRTWDOG_InterruptFlag);
*
```

Parameters

|             |                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                    |
| <i>mask</i> | The status flags to clear. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kRTWDOG_InterruptFlag</li> </ul> |

**36.6.10 static void RTWDOG\_SetTimeoutValue ( RTWDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]**

This function writes a timeout value into the WDOG\_TOVAL register. The WDOG\_TOVAL register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

## Function Documentation

Parameters

|                     |                                                    |
|---------------------|----------------------------------------------------|
| <i>base</i>         | RTWDOG peripheral base address                     |
| <i>timeoutCount</i> | RTWDOG timeout value, count of RTWDOG clock ticks. |

### 36.6.11 static void RTWDOG\_SetWindowValue ( RTWDOG\_Type \* *base*, uint16\_t *windowValue* ) [inline], [static]

This function writes a window value into the WDOG\_WIN register. The WDOG\_WIN register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | RTWDOG peripheral base address. |
| <i>windowValue</i> | RTWDOG window value.            |

### 36.6.12 static void RTWDOG\_Unlock ( RTWDOG\_Type \* *base* ) [inline], [static]

This function unlocks the RTWDOG register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

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

### 36.6.13 static void RTWDOG\_Refresh ( RTWDOG\_Type \* *base* ) [inline], [static]

This function feeds the RTWDOG. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

Parameters

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

**36.6.14** `static uint16_t RTWDOG_GetCounterValue ( RTWDOG_Type * base )`  
`[inline], [static]`

This function gets the RTWDOG counter value.

Parameters

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

Returns

Current RTWDOG counter value.



## Chapter 37

# QTMR: Quad Timer Driver

### 37.1 Overview

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

### Data Structures

- struct `qtmr_config_t`  
*Quad Timer config structure. [More...](#)*

### Enumerations

- enum `qtmr_primary_count_source_t` {  
    `kQTMR_ClockCounter0InputPin` = 0,  
    `kQTMR_ClockCounter1InputPin`,  
    `kQTMR_ClockCounter2InputPin`,  
    `kQTMR_ClockCounter3InputPin`,  
    `kQTMR_ClockCounter0Output`,  
    `kQTMR_ClockCounter1Output`,  
    `kQTMR_ClockCounter2Output`,  
    `kQTMR_ClockCounter3Output`,  
    `kQTMR_ClockDivide_1`,  
    `kQTMR_ClockDivide_2`,  
    `kQTMR_ClockDivide_4`,  
    `kQTMR_ClockDivide_8`,  
    `kQTMR_ClockDivide_16`,  
    `kQTMR_ClockDivide_32`,  
    `kQTMR_ClockDivide_64`,  
    `kQTMR_ClockDivide_128` }  
*Quad Timer primary clock source selection.*
- enum `qtmr_input_source_t` {  
    `kQTMR_Counter0InputPin` = 0,  
    `kQTMR_Counter1InputPin`,  
    `kQTMR_Counter2InputPin`,  
    `kQTMR_Counter3InputPin` }  
*Quad Timer input sources selection.*
- enum `qtmr_counting_mode_t` {

## Overview

```
kQTMR_NoOperation = 0,  
kQTMR_PriSrcRiseEdge,  
kQTMR_PriSrcRiseAndFallEdge,  
kQTMR_PriSrcRiseEdgeSecInpHigh,  
kQTMR_QuadCountMode,  
kQTMR_PriSrcRiseEdgeSecDir,  
kQTMR_SecSrcTrigPriCnt,  
kQTMR_CascadeCount }
```

*Quad Timer counting mode selection.*

- enum qtmr\_output\_mode\_t {  
kQTMR\_AssertWhenCountActive = 0,  
kQTMR\_ClearOnCompare,  
kQTMR\_SetOnCompare,  
kQTMR\_ToggleOnCompare,  
kQTMR\_ToggleOnAltCompareReg,  
kQTMR\_SetOnCompareClearOnSecSrcInp,  
kQTMR\_SetOnCompareClearOnCountRoll,  
kQTMR\_EnableGateClock }

*Quad Timer output mode selection.*

- enum qtmr\_input\_capture\_edge\_t {  
kQTMR\_NoCapture = 0,  
kQTMR\_RisingEdge,  
kQTMR\_FallingEdge,  
kQTMR\_RisingAndFallingEdge }

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum qtmr\_preload\_control\_t {  
kQTMR\_NoPreload = 0,  
kQTMR\_LoadOnComp1,  
kQTMR\_LoadOnComp2 }

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum qtmr\_debug\_action\_t {  
kQTMR\_RunNormalInDebug = 0U,  
kQTMR\_HaltCounter,  
kQTMR\_ForceOutToZero,  
kQTMR\_HaltCountForceOutZero }

*List of Quad Timer run options when in Debug mode.*

- enum qtmr\_interrupt\_enable\_t {  
kQTMR\_CompareInterruptEnable = (1U << 0),  
kQTMR\_Compare1InterruptEnable = (1U << 1),  
kQTMR\_Compare2InterruptEnable = (1U << 2),  
kQTMR\_OverflowInterruptEnable = (1U << 3),  
kQTMR\_EdgeInterruptEnable = (1U << 4) }

*List of Quad Timer interrupts.*

- enum qtmr\_status\_flags\_t {

```

kQTMR_CompareFlag = (1U << 0),
kQTMR_Compare1Flag = (1U << 1),
kQTMR_Compare2Flag = (1U << 2),
kQTMR_OverflowFlag = (1U << 3),
kQTMR_EdgeFlag = (1U << 4) }

```

*List of Quad Timer flags.*

## Functions

- status\_t [QTMR\\_SetupPwm](#) (TMR\_Type \*base, uint32\_t pwmFreqHz, uint8\_t dutyCyclePercent, bool outputPolarity, uint32\_t srcClock\_Hz)  
*Sets up Quad timer module for PWM signal output.*
- void [QTMR\\_SetupInputCapture](#) (TMR\_Type \*base, [qtmr\\_input\\_source\\_t](#) capturePin, bool inputPolarity, bool reloadOnCapture, [qtmr\\_input\\_capture\\_edge\\_t](#) captureMode)  
*Allows the user to count the source clock cycles until a capture event arrives.*

## Driver version

- #define [FSL\\_QTMR\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void [QTMR\\_Init](#) (TMR\_Type \*base, const [qtmr\\_config\\_t](#) \*config)  
*Ungates the Quad Timer clock and configures the peripheral for basic operation.*
- void [QTMR\\_Deinit](#) (TMR\_Type \*base)  
*Stops the counter and gates the Quad Timer clock.*
- void [QTMR\\_GetDefaultConfig](#) ([qtmr\\_config\\_t](#) \*config)  
*Fill in the Quad Timer config struct with the default settings.*

## Interrupt Interface

- void [QTMR\\_EnableInterrupts](#) (TMR\_Type \*base, uint32\_t mask)  
*Enables the selected Quad Timer interrupts.*
- void [QTMR\\_DisableInterrupts](#) (TMR\_Type \*base, uint32\_t mask)  
*Disables the selected Quad Timer interrupts.*
- uint32\_t [QTMR\\_GetEnabledInterrupts](#) (TMR\_Type \*base)  
*Gets the enabled Quad Timer interrupts.*

## Status Interface

- uint32\_t [QTMR\\_GetStatus](#) (TMR\_Type \*base)  
*Gets the Quad Timer status flags.*
- void [QTMR\\_ClearStatusFlags](#) (TMR\_Type \*base, uint32\_t mask)  
*Clears the Quad Timer status flags.*

## Read and Write the timer period

- void [QTMR\\_SetTimerPeriod](#) (TMR\_Type \*base, uint16\_t ticks)

## Enumeration Type Documentation

- Sets the timer period in ticks.*
- static uint16\_t [QTMR\\_GetCurrentTimerCount](#) (TMR\_Type \*base)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [QTMR\\_StartTimer](#) (TMR\_Type \*base, [qtmr\\_counting\\_mode\\_t](#) clockSource)  
*Starts the Quad Timer counter.*
- static void [QTMR\\_StopTimer](#) (TMR\_Type \*base)  
*Stops the Quad Timer counter.*

## 37.2 Data Structure Documentation

### 37.2.1 struct qtmr\_config\_t

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the [QTMR\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

### Data Fields

- [qtmr\\_primary\\_count\\_source\\_t](#) primarySource  
*Specify the primary count source.*
- [qtmr\\_input\\_source\\_t](#) secondarySource  
*Specify the secondary count source.*
- bool [enableMasterMode](#)  
*true: Broadcast compare function output to other counters; false no broadcast*
- bool [enableExternalForce](#)  
*true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter*
- uint8\_t [faultFilterCount](#)  
*Fault filter count.*
- uint8\_t [faultFilterPeriod](#)  
*Fault filter period; value of 0 will bypass the filter.*
- [qtmr\\_debug\\_action\\_t](#) debugMode  
*Operation in Debug mode.*

## 37.3 Enumeration Type Documentation

### 37.3.1 enum qtmr\_primary\_count\_source\_t

Enumerator

- [kQTMR\\_ClockCounter0InputPin](#) Use counter 0 input pin.
- [kQTMR\\_ClockCounter1InputPin](#) Use counter 1 input pin.
- [kQTMR\\_ClockCounter2InputPin](#) Use counter 2 input pin.



***kQTMR\_ClockCounter3InputPin*** Use counter 3 input pin.  
***kQTMR\_ClockCounter0Output*** Use counter 0 output.  
***kQTMR\_ClockCounter1Output*** Use counter 1 output.  
***kQTMR\_ClockCounter2Output*** Use counter 2 output.  
***kQTMR\_ClockCounter3Output*** Use counter 3 output.  
***kQTMR\_ClockDivide\_1*** IP bus clock divide by 1 prescaler.  
***kQTMR\_ClockDivide\_2*** IP bus clock divide by 2 prescaler.  
***kQTMR\_ClockDivide\_4*** IP bus clock divide by 4 prescaler.  
***kQTMR\_ClockDivide\_8*** IP bus clock divide by 8 prescaler.  
***kQTMR\_ClockDivide\_16*** IP bus clock divide by 16 prescaler.  
***kQTMR\_ClockDivide\_32*** IP bus clock divide by 32 prescaler.  
***kQTMR\_ClockDivide\_64*** IP bus clock divide by 64 prescaler.  
***kQTMR\_ClockDivide\_128*** IP bus clock divide by 128 prescaler.

### 37.3.2 enum qtmr\_input\_source\_t

Enumerator

***kQTMR\_Counter0InputPin*** Use counter 0 input pin.  
***kQTMR\_Counter1InputPin*** Use counter 1 input pin.  
***kQTMR\_Counter2InputPin*** Use counter 2 input pin.  
***kQTMR\_Counter3InputPin*** Use counter 3 input pin.

### 37.3.3 enum qtmr\_counting\_mode\_t

Enumerator

***kQTMR\_NoOperation*** No operation.  
***kQTMR\_PriSrcRiseEdge*** Count rising edges or primary source.  
***kQTMR\_PriSrcRiseAndFallEdge*** Count rising and falling edges of primary source.  
***kQTMR\_PriSrcRiseEdgeSecInpHigh*** Count rise edges of pri SRC while sec inp high active.  
***kQTMR\_QuadCountMode*** Quadrature count mode, uses pri and sec sources.  
***kQTMR\_PriSrcRiseEdgeSecDir*** Count rising edges of pri SRC; sec SRC specifies dir.  
***kQTMR\_SecSrcTrigPriCnt*** Edge of sec SRC trigger primary count until compare.  
***kQTMR\_CascadeCount*** Cascaded count mode (up/down)

### 37.3.4 enum qtmr\_output\_mode\_t

Enumerator

***kQTMR\_AssertWhenCountActive*** Assert OFLAG while counter is active.

## Enumeration Type Documentation

*kQTMR\_ClearOnCompare* Clear OFLAG on successful compare.  
*kQTMR\_SetOnCompare* Set OFLAG on successful compare.  
*kQTMR\_ToggleOnCompare* Toggle OFLAG on successful compare.  
*kQTMR\_ToggleOnAltCompareReg* Toggle OFLAG using alternating compare registers.  
*kQTMR\_SetOnCompareClearOnSecSrcInp* Set OFLAG on compare, clear on sec SRC input edge.  
  
*kQTMR\_SetOnCompareClearOnCountRoll* Set OFLAG on compare, clear on counter rollover.  
*kQTMR\_EnableGateClock* Enable gated clock output while count is active.

### 37.3.5 enum qtmr\_input\_capture\_edge\_t

Enumerator

*kQTMR\_NoCapture* Capture is disabled.  
*kQTMR\_RisingEdge* Capture on rising edge (IPS=0) or falling edge (IPS=1)  
*kQTMR\_FallingEdge* Capture on falling edge (IPS=0) or rising edge (IPS=1)  
*kQTMR\_RisingAndFallingEdge* Capture on both edges.

### 37.3.6 enum qtmr\_preload\_control\_t

Enumerator

*kQTMR\_NoPreload* Never preload.  
*kQTMR\_LoadOnComp1* Load upon successful compare with value in COMP1.  
*kQTMR\_LoadOnComp2* Load upon successful compare with value in COMP2.

### 37.3.7 enum qtmr\_debug\_action\_t

Enumerator

*kQTMR\_RunNormalInDebug* Continue with normal operation.  
*kQTMR\_HaltCounter* Halt counter.  
*kQTMR\_ForceOutToZero* Force output to logic 0.  
*kQTMR\_HaltCountForceOutZero* Halt counter and force output to logic 0.

### 37.3.8 enum qtmr\_interrupt\_enable\_t

Enumerator

*kQTMR\_CompareInterruptEnable* Compare interrupt.

*kQTMR\_Compare1InterruptEnable* Compare 1 interrupt.  
*kQTMR\_Compare2InterruptEnable* Compare 2 interrupt.  
*kQTMR\_OverflowInterruptEnable* Timer overflow interrupt.  
*kQTMR\_EdgeInterruptEnable* Input edge interrupt.

### 37.3.9 enum qtmr\_status\_flags\_t

Enumerator

*kQTMR\_CompareFlag* Compare flag.  
*kQTMR\_Compare1Flag* Compare 1 flag.  
*kQTMR\_Compare2Flag* Compare 2 flag.  
*kQTMR\_OverflowFlag* Timer overflow flag.  
*kQTMR\_EdgeFlag* Input edge flag.

## 37.4 Function Documentation

### 37.4.1 void QTMR\_Init ( TMR\_Type \* *base*, const qtmr\_config\_t \* *config* )

Note

This API should be called at the beginning of the application using the Quad Timer driver.

Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | Quad Timer peripheral base address            |
| <i>config</i> | Pointer to user's Quad Timer config structure |

### 37.4.2 void QTMR\_Deinit ( TMR\_Type \* *base* )

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | Quad Timer peripheral base address |
|-------------|------------------------------------|

### 37.4.3 void QTMR\_GetDefaultConfig ( qtmr\_config\_t \* *config* )

The default values are:

```
* config->debugMode = kQTMR_RunNormalInDebug;
* config->enableExternalForce = false;
* config->enableMasterMode = false;
```

## Function Documentation

```
* config->faultFilterCount = 0;  
* config->faultFilterPeriod = 0;  
* config->primarySource = kQTMR_ClockDivide_2;  
* config->secondarySource = kQTMR_Counter0InputPin;  
*
```

### Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>config</i> | Pointer to user's Quad Timer config structure. |
|---------------|------------------------------------------------|

### 37.4.4 **status\_t QTMR\_SetupPwm ( TMR\_Type \* *base*, uint32\_t *pwmFreqHz*, uint8\_t *dutyCyclePercent*, bool *outputPolarity*, uint32\_t *srcClock\_Hz* )**

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

### Parameters

|                          |                                                                                                                           |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | Quad Timer peripheral base address                                                                                        |
| <i>pwmFreqHz</i>         | PWM signal frequency in Hz                                                                                                |
| <i>dutyCycle-Percent</i> | PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |
| <i>outputPolarity</i>    | true: invert polarity of the output signal, false: no inversion                                                           |
| <i>srcClock_Hz</i>       | Main counter clock in Hz.                                                                                                 |

### Returns

Returns an error if there was error setting up the signal.

### 37.4.5 **void QTMR\_SetupInputCapture ( TMR\_Type \* *base*, qtmr\_input\_source\_t *capturePin*, bool *inputPolarity*, bool *reloadOnCapture*, qtmr\_input\_capture\_edge\_t *captureMode* )**

The count is stored in the capture register.

### Parameters

|                         |                                                                         |
|-------------------------|-------------------------------------------------------------------------|
| <i>base</i>             | Quad Timer peripheral base address                                      |
| <i>capturePin</i>       | Pin through which we receive the input signal to trigger the capture    |
| <i>inputPolarity</i>    | true: invert polarity of the input signal, false: no inversion          |
| <i>reloadOn-Capture</i> | true: reload the counter when an input capture occurs, false: no reload |
| <i>captureMode</i>      | Specifies which edge of the input signal triggers a capture             |

#### 37.4.6 void QTMR\_EnableInterrupts ( TMR\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Quad Timer peripheral base address                                                                                   |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a> |

#### 37.4.7 void QTMR\_DisableInterrupts ( TMR\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Quad Timer peripheral base address                                                                                   |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a> |

#### 37.4.8 uint32\_t QTMR\_GetEnabledInterrupts ( TMR\_Type \* *base* )

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | Quad Timer peripheral base address |
|-------------|------------------------------------|

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [qtmr\\_interrupt\\_enable\\_t](#)

#### 37.4.9 uint32\_t QTMR\_GetStatus ( TMR\_Type \* *base* )

## Function Documentation

### Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | Quad Timer peripheral base address |
|-------------|------------------------------------|

### Returns

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

### 37.4.10 void QTMR\_ClearStatusFlags ( TMR\_Type \* *base*, uint32\_t *mask* )

### Parameters

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

### 37.4.11 void QTMR\_SetTimerPeriod ( TMR\_Type \* *base*, uint16\_t *ticks* )

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

### Note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in `fsl_common.h` to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.

### Parameters

|              |                                    |
|--------------|------------------------------------|
| <i>base</i>  | Quad Timer peripheral base address |
| <i>ticks</i> | Timer period in units of ticks     |

### 37.4.12 static uint16\_t QTMR\_GetCurrentTimerCount ( TMR\_Type \* *base* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

## Note

User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | Quad Timer peripheral base address |
|-------------|------------------------------------|

## Returns

Current counter value in ticks

**37.4.13** `static void QTMR_StartTimer ( TMR_Type * base, qtmr_counting_mode_t clockSource ) [inline], [static]`

## Parameters

|                    |                                    |
|--------------------|------------------------------------|
| <i>base</i>        | Quad Timer peripheral base address |
| <i>clockSource</i> | Quad Timer clock source            |

**37.4.14** `static void QTMR_StopTimer ( TMR_Type * base ) [inline], [static]`

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | Quad Timer peripheral base address |
|-------------|------------------------------------|





## Chapter 38

# SAI: Serial Audio Interface

### 38.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Audio Interface (SAI) module of MCUXpresso SDK devices.

SAI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SAI initialization, configuration and operation, and for optimization and customization purposes. Using the functional API requires the knowledge of the SAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SAI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `sai_handle_t` as the first parameter. Initialize the handle by calling the [SAI\\_TransferTxCreateHandle\(\)](#) or [SAI\\_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SAI\\_TransferSendNonBlocking\(\)](#) and [SAI\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SAI_TxIdle` and `kStatus_SAI_RxIdle` status.

### 38.2 Typical use case

#### 38.2.1 SAI Send/receive using an interrupt method

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

#### 38.2.2 SAI Send/receive using a DMA method

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

### Modules

- [SAI DMA Driver](#)
- [SAI SDMA Driver](#)
- [SAI eDMA Driver](#)

## Typical use case

## Data Structures

- struct `sai_config_t`  
*SAI user configuration structure. [More...](#)*
- struct `sai_transfer_format_t`  
*sai transfer format [More...](#)*
- struct `sai_transfer_t`  
*SAI transfer structure. [More...](#)*
- struct `sai_handle_t`  
*SAI handle structure. [More...](#)*

## Macros

- #define `SAI_XFER_QUEUE_SIZE` (4)  
*SAI transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* `sai_transfer_callback_t`)(I2S\_Type \*base, sai\_handle\_t \*handle, status\_t status, void \*userData)  
*SAI transfer callback prototype.*

## Enumerations

- enum `_sai_status_t` {  
`kStatus_SAI_TxBusy` = MAKE\_STATUS(kStatusGroup\_SAI, 0),  
`kStatus_SAI_RxBusy` = MAKE\_STATUS(kStatusGroup\_SAI, 1),  
`kStatus_SAI_TxError` = MAKE\_STATUS(kStatusGroup\_SAI, 2),  
`kStatus_SAI_RxError` = MAKE\_STATUS(kStatusGroup\_SAI, 3),  
`kStatus_SAI_QueueFull` = MAKE\_STATUS(kStatusGroup\_SAI, 4),  
`kStatus_SAI_TxIdle` = MAKE\_STATUS(kStatusGroup\_SAI, 5),  
`kStatus_SAI_RxIdle` = MAKE\_STATUS(kStatusGroup\_SAI, 6) }  
*SAI return status.*
- enum `sai_protocol_t` {  
`kSAI_BusLeftJustified` = 0x0U,  
`kSAI_BusRightJustified`,  
`kSAI_BusI2S`,  
`kSAI_BusPCMA`,  
`kSAI_BusPCMB` }  
*Define the SAI bus type.*
- enum `sai_master_slave_t` {  
`kSAI_Master` = 0x0U,  
`kSAI_Slave` = 0x1U }  
*Master or slave mode.*
- enum `sai_mono_stereo_t` {  
`kSAI_Stereo` = 0x0U,  
`kSAI_MonoRight`,  
`kSAI_MonoLeft` }

- Mono or stereo audio format.*

  - enum `sai_data_order_t` {  
`kSAI_DataLSB = 0x0U`,  
`kSAI_DataMSB` }

*SAI data order, MSB or LSB.*
- enum `sai_clock_polarity_t` {  
`kSAI_PolarityActiveHigh = 0x0U`,  
`kSAI_PolarityActiveLow` }

*SAI clock polarity, active high or low.*
- enum `sai_sync_mode_t` {  
`kSAI_ModeAsync = 0x0U`,  
`kSAI_ModeSync`,  
`kSAI_ModeSyncWithOtherTx`,  
`kSAI_ModeSyncWithOtherRx` }

*Synchronous or asynchronous mode.*
- enum `sai_mclk_source_t` {  
`kSAI_MclkSourceSysclk = 0x0U`,  
`kSAI_MclkSourceSelect1`,  
`kSAI_MclkSourceSelect2`,  
`kSAI_MclkSourceSelect3` }

*Master clock source.*
- enum `sai_bclk_source_t` {  
`kSAI_BclkSourceBusclk = 0x0U`,  
`kSAI_BclkSourceMclkDiv`,  
`kSAI_BclkSourceOtherSai0`,  
`kSAI_BclkSourceOtherSai1` }

*Bit clock source.*
- enum `_sai_interrupt_enable_t` {  
`kSAI_WordStartInterruptEnable`,  
`kSAI_SyncErrorInterruptEnable = I2S_TCSR_SEIE_MASK`,  
`kSAI_FIFOWarningInterruptEnable = I2S_TCSR_FWIE_MASK`,  
`kSAI_FIFOErrorInterruptEnable = I2S_TCSR_FEIE_MASK`,  
`kSAI_FIFORequestInterruptEnable = I2S_TCSR_FRIE_MASK` }

*The SAI interrupt enable flag.*
- enum `_sai_dma_enable_t` {  
`kSAI_FIFOWarningDMAEnable = I2S_TCSR_FWDE_MASK`,  
`kSAI_FIFORequestDMAEnable = I2S_TCSR_FRDE_MASK` }

*The DMA request sources.*
- enum `_sai_flags` {  
`kSAI_WordStartFlag = I2S_TCSR_WSF_MASK`,  
`kSAI_SyncErrorFlag = I2S_TCSR_SEF_MASK`,  
`kSAI_FIFOErrorFlag = I2S_TCSR_FEF_MASK`,  
`kSAI_FIFORequestFlag = I2S_TCSR_FRF_MASK`,  
`kSAI_FIFOWarningFlag = I2S_TCSR_FWF_MASK` }

*The SAI status flag.*
- enum `sai_reset_type_t` {

## Typical use case

```
kSAI_ResetTypeSoftware = I2S_TCSR_SR_MASK,  
kSAI_ResetTypeFIFO = I2S_TCSR_FR_MASK,  
kSAI_ResetAll = I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK }
```

*The reset type.*

- enum `sai_fifo_packing_t` {  
    `kSAI_FifoPackingDisabled` = 0x0U,  
    `kSAI_FifoPacking8bit` = 0x2U,  
    `kSAI_FifoPacking16bit` = 0x3U }

*The SAI packing mode The mode includes 8 bit and 16 bit packing.*

- enum `sai_sample_rate_t` {  
    `kSAI_SampleRate8KHz` = 8000U,  
    `kSAI_SampleRate11025Hz` = 11025U,  
    `kSAI_SampleRate12KHz` = 12000U,  
    `kSAI_SampleRate16KHz` = 16000U,  
    `kSAI_SampleRate22050Hz` = 22050U,  
    `kSAI_SampleRate24KHz` = 24000U,  
    `kSAI_SampleRate32KHz` = 32000U,  
    `kSAI_SampleRate44100Hz` = 44100U,  
    `kSAI_SampleRate48KHz` = 48000U,  
    `kSAI_SampleRate96KHz` = 96000U }

*Audio sample rate.*

- enum `sai_word_width_t` {  
    `kSAI_WordWidth8bits` = 8U,  
    `kSAI_WordWidth16bits` = 16U,  
    `kSAI_WordWidth24bits` = 24U,  
    `kSAI_WordWidth32bits` = 32U }

*Audio word width.*

## Driver version

- #define `FSL_SAI_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 4))  
*Version 2.1.4.*

## Initialization and deinitialization

- void `SAI_TxInit` (I2S\_Type \*base, const `sai_config_t` \*config)  
*Initializes the SAI Tx peripheral.*
- void `SAI_RxInit` (I2S\_Type \*base, const `sai_config_t` \*config)  
*Initializes the SAI Rx peripheral.*
- void `SAI_TxGetDefaultConfig` (`sai_config_t` \*config)  
*Sets the SAI Tx configuration structure to default values.*
- void `SAI_RxGetDefaultConfig` (`sai_config_t` \*config)  
*Sets the SAI Rx configuration structure to default values.*
- void `SAI_Deinit` (I2S\_Type \*base)  
*De-initializes the SAI peripheral.*
- void `SAI_TxReset` (I2S\_Type \*base)  
*Resets the SAI Tx.*
- void `SAI_RxReset` (I2S\_Type \*base)

- *Resets the SAI Rx.*
- void [SAI\\_TxEnable](#) (I2S\_Type \*base, bool enable)  
*Enables/disables the SAI Tx.*
- void [SAI\\_RxEnable](#) (I2S\_Type \*base, bool enable)  
*Enables/disables the SAI Rx.*

## Status

- static uint32\_t [SAI\\_TxGetStatusFlag](#) (I2S\_Type \*base)  
*Gets the SAI Tx status flag state.*
- static void [SAI\\_TxClearStatusFlags](#) (I2S\_Type \*base, uint32\_t mask)  
*Clears the SAI Tx status flag state.*
- static uint32\_t [SAI\\_RxGetStatusFlag](#) (I2S\_Type \*base)  
*Gets the SAI Rx status flag state.*
- static void [SAI\\_RxClearStatusFlags](#) (I2S\_Type \*base, uint32\_t mask)  
*Clears the SAI Rx status flag state.*
- void [SAI\\_TxSoftwareReset](#) (I2S\_Type \*base, sai\_reset\_type\_t type)  
*Do software reset or FIFO reset .*
- void [SAI\\_RxSoftwareReset](#) (I2S\_Type \*base, sai\_reset\_type\_t type)  
*Do software reset or FIFO reset .*
- void [SAI\\_TxSetChannelFIFOMask](#) (I2S\_Type \*base, uint8\_t mask)  
*Set the Tx channel FIFO enable mask.*
- void [SAI\\_RxSetChannelFIFOMask](#) (I2S\_Type \*base, uint8\_t mask)  
*Set the Rx channel FIFO enable mask.*
- void [SAI\\_TxSetDataOrder](#) (I2S\_Type \*base, sai\_data\_order\_t order)  
*Set the Tx data order.*
- void [SAI\\_RxSetDataOrder](#) (I2S\_Type \*base, sai\_data\_order\_t order)  
*Set the Rx data order.*
- void [SAI\\_TxSetBitClockPolarity](#) (I2S\_Type \*base, sai\_clock\_polarity\_t polarity)  
*Set the Tx data order.*
- void [SAI\\_RxSetBitClockPolarity](#) (I2S\_Type \*base, sai\_clock\_polarity\_t polarity)  
*Set the Rx data order.*
- void [SAI\\_TxSetFrameSyncPolarity](#) (I2S\_Type \*base, sai\_clock\_polarity\_t polarity)  
*Set the Tx data order.*
- void [SAI\\_RxSetFrameSyncPolarity](#) (I2S\_Type \*base, sai\_clock\_polarity\_t polarity)  
*Set the Rx data order.*
- void [SAI\\_TxSetFIFOPacking](#) (I2S\_Type \*base, sai\_fifo\_packing\_t pack)  
*Set Tx FIFO packing feature.*
- void [SAI\\_RxSetFIFOPacking](#) (I2S\_Type \*base, sai\_fifo\_packing\_t pack)  
*Set Rx FIFO packing feature.*
- static void [SAI\\_TxSetFIFOErrorContinue](#) (I2S\_Type \*base, bool isEnabled)  
*Set Tx FIFO error continue.*
- static void [SAI\\_RxSetFIFOErrorContinue](#) (I2S\_Type \*base, bool isEnabled)  
*Set Rx FIFO error continue.*

## Interrupts

- static void [SAI\\_TxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables the SAI Tx interrupt requests.*
- static void [SAI\\_RxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables the SAI Rx interrupt requests.*

## Typical use case

- static void [SAI\\_TxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables the SAI Tx interrupt requests.*
- static void [SAI\\_RxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables the SAI Rx interrupt requests.*

## DMA Control

- static void [SAI\\_TxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the SAI Tx DMA requests.*
- static void [SAI\\_RxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the SAI Rx DMA requests.*
- static uint32\_t [SAI\\_TxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Tx data register address.*
- static uint32\_t [SAI\\_RxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Rx data register address.*

## Bus Operations

- void [SAI\\_TxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_RxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- void [SAI\\_WriteBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Sends data using a blocking method.*
- static void [SAI\\_WriteData](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t data)  
*Writes data into SAI FIFO.*
- void [SAI\\_ReadBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Receives data using a blocking method.*
- static uint32\_t [SAI\\_ReadData](#) (I2S\_Type \*base, uint32\_t channel)  
*Reads data from the SAI FIFO.*

## Transactional

- void [SAI\\_TransferTxCreateHandle](#) (I2S\_Type \*base, [sai\\_handle\\_t](#) \*handle, [sai\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SAI Tx handle.*
- void [SAI\\_TransferRxCreateHandle](#) (I2S\_Type \*base, [sai\\_handle\\_t](#) \*handle, [sai\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SAI Rx handle.*
- status\_t [SAI\\_TransferTxSetFormat](#) (I2S\_Type \*base, [sai\\_handle\\_t](#) \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- status\_t [SAI\\_TransferRxSetFormat](#) (I2S\_Type \*base, [sai\\_handle\\_t](#) \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*

- status\_t [SAI\\_TransferSendNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking send transfer on SAI.*
- status\_t [SAI\\_TransferReceiveNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking receive transfer on SAI.*
- status\_t [SAI\\_TransferGetSendCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a set byte count.*
- status\_t [SAI\\_TransferGetReceiveCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a received byte count.*
- void [SAI\\_TransferAbortSend](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Aborts the current send.*
- void [SAI\\_TransferAbortReceive](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Aborts the current IRQ receive.*
- void [SAI\\_TransferTerminateSend](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Terminate all SAI send.*
- void [SAI\\_TransferTerminateReceive](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Terminate all SAI receive.*
- void [SAI\\_TransferTxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Tx interrupt handler.*
- void [SAI\\_TransferRxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Tx interrupt handler.*

## 38.3 Data Structure Documentation

### 38.3.1 struct sai\_config\_t

#### Data Fields

- [sai\\_protocol\\_t protocol](#)  
*Audio bus protocol in SAI.*
- [sai\\_sync\\_mode\\_t syncMode](#)  
*SAI sync mode, control Tx/Rx clock sync.*
- [sai\\_mclk\\_source\\_t mclkSource](#)  
*Master Clock source.*
- [sai\\_bclk\\_source\\_t bclkSource](#)  
*Bit Clock source.*
- [sai\\_master\\_slave\\_t masterSlave](#)  
*Master or slave.*

### 38.3.2 struct sai\_transfer\_format\_t

#### Data Fields

- uint32\_t [sampleRate\\_Hz](#)  
*Sample rate of audio data.*
- uint32\_t [bitWidth](#)  
*Data length of audio data, usually 8/16/24/32 bits.*

## Data Structure Documentation

- [sai\\_mono\\_stereo\\_t stereo](#)  
*Mono or stereo.*
- [uint32\\_t masterClockHz](#)  
*Master clock frequency in Hz.*
- [uint8\\_t watermark](#)  
*Watermark value.*
- [uint8\\_t channel](#)  
*Data channel used in transfer.*
- [sai\\_protocol\\_t protocol](#)  
*Which audio protocol used.*
- [bool isFrameSyncCompact](#)  
*True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.*

### 38.3.2.0.0.38 Field Documentation

38.3.2.0.0.38.1 [uint8\\_t sai\\_transfer\\_format\\_t::channel](#)

38.3.2.0.0.38.2 [bool sai\\_transfer\\_format\\_t::isFrameSyncCompact](#)

### 38.3.3 struct sai\_transfer\_t

#### Data Fields

- [uint8\\_t \\* data](#)  
*Data start address to transfer.*
- [size\\_t dataSize](#)  
*Transfer size.*

### 38.3.3.0.0.39 Field Documentation

38.3.3.0.0.39.1 [uint8\\_t\\* sai\\_transfer\\_t::data](#)

38.3.3.0.0.39.2 [size\\_t sai\\_transfer\\_t::dataSize](#)

### 38.3.4 struct \_sai\_handle

#### Data Fields

- [uint32\\_t state](#)  
*Transfer status.*
- [sai\\_transfer\\_callback\\_t callback](#)  
*Callback function called at transfer event.*
- [void \\* userData](#)  
*Callback parameter passed to callback function.*
- [uint8\\_t bitWidth](#)  
*Bit width for transfer, 8/16/24/32 bits.*
- [uint8\\_t channel](#)  
*Transfer channel.*



- `sai_transfer_t saiQueue` [`SAI_XFER_QUEUE_SIZE`]  
*Transfer queue storing queued transfer.*
- `size_t transferSize` [`SAI_XFER_QUEUE_SIZE`]  
*Data bytes need to transfer.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*
- `uint8_t watermark`  
*Watermark value.*

### 38.4 Macro Definition Documentation

#### 38.4.1 #define SAI\_XFER\_QUEUE\_SIZE (4)

### 38.5 Enumeration Type Documentation

#### 38.5.1 enum \_sai\_status\_t

Enumerator

*kStatus\_SAI\_TxBusy* SAI Tx is busy.  
*kStatus\_SAI\_RxBusy* SAI Rx is busy.  
*kStatus\_SAI\_TxError* SAI Tx FIFO error.  
*kStatus\_SAI\_RxError* SAI Rx FIFO error.  
*kStatus\_SAI\_QueueFull* SAI transfer queue is full.  
*kStatus\_SAI\_TxIdle* SAI Tx is idle.  
*kStatus\_SAI\_RxIdle* SAI Rx is idle.

#### 38.5.2 enum sai\_protocol\_t

Enumerator

*kSAI\_BusLeftJustified* Uses left justified format.  
*kSAI\_BusRightJustified* Uses right justified format.  
*kSAI\_BusI2S* Uses I2S format.  
*kSAI\_BusPCMA* Uses I2S PCM A format.  
*kSAI\_BusPCMB* Uses I2S PCM B format.

#### 38.5.3 enum sai\_master\_slave\_t

Enumerator

*kSAI\_Master* Master mode.  
*kSAI\_Slave* Slave mode.

## Enumeration Type Documentation

### 38.5.4 enum sai\_mono\_stereo\_t

Enumerator

- kSAI\_Stereo* Stereo sound.
- kSAI\_MonoRight* Only Right channel have sound.
- kSAI\_MonoLeft* Only left channel have sound.

### 38.5.5 enum sai\_data\_order\_t

Enumerator

- kSAI\_DataLSB* LSB bit transferred first.
- kSAI\_DataMSB* MSB bit transferred first.

### 38.5.6 enum sai\_clock\_polarity\_t

Enumerator

- kSAI\_PolarityActiveHigh* Clock active high.
- kSAI\_PolarityActiveLow* Clock active low.

### 38.5.7 enum sai\_sync\_mode\_t

Enumerator

- kSAI\_ModeAsync* Asynchronous mode.
- kSAI\_ModeSync* Synchronous mode (with receiver or transmit)
- kSAI\_ModeSyncWithOtherTx* Synchronous with another SAI transmit.
- kSAI\_ModeSyncWithOtherRx* Synchronous with another SAI receiver.

### 38.5.8 enum sai\_mclk\_source\_t

Enumerator

- kSAI\_MclkSourceSysclk* Master clock from the system clock.
- kSAI\_MclkSourceSelect1* Master clock from source 1.
- kSAI\_MclkSourceSelect2* Master clock from source 2.
- kSAI\_MclkSourceSelect3* Master clock from source 3.

### 38.5.9 enum sai\_bclk\_source\_t

Enumerator

- kSAI\_BclkSourceBusclk* Bit clock using bus clock.
- kSAI\_BclkSourceMclkDiv* Bit clock using master clock divider.
- kSAI\_BclkSourceOtherSai0* Bit clock from other SAI device.
- kSAI\_BclkSourceOtherSai1* Bit clock from other SAI device.

### 38.5.10 enum \_sai\_interrupt\_enable\_t

Enumerator

- kSAI\_WordStartInterruptEnable* Word start flag, means the first word in a frame detected.
- kSAI\_SyncErrorInterruptEnable* Sync error flag, means the sync error is detected.
- kSAI\_FIFOWarningInterruptEnable* FIFO warning flag, means the FIFO is empty.
- kSAI\_FIFOErrorInterruptEnable* FIFO error flag.
- kSAI\_FIFORequestInterruptEnable* FIFO request, means reached watermark.

### 38.5.11 enum \_sai\_dma\_enable\_t

Enumerator

- kSAI\_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.
- kSAI\_FIFORequestDMAEnable* FIFO request caused by the DMA request.

### 38.5.12 enum \_sai\_flags

Enumerator

- kSAI\_WordStartFlag* Word start flag, means the first word in a frame detected.
- kSAI\_SyncErrorFlag* Sync error flag, means the sync error is detected.
- kSAI\_FIFOErrorFlag* FIFO error flag.
- kSAI\_FIFORequestFlag* FIFO request flag.
- kSAI\_FIFOWarningFlag* FIFO warning flag.

### 38.5.13 enum sai\_reset\_type\_t

Enumerator

- kSAI\_ResetTypeSoftware* Software reset, reset the logic state.

## Function Documentation

*kSAI\_ResetTypeFIFO* FIFO reset, reset the FIFO read and write pointer.  
*kSAI\_ResetAll* All reset.

### 38.5.14 enum sai\_fifo\_packing\_t

Enumerator

*kSAI\_FifoPackingDisabled* Packing disabled.  
*kSAI\_FifoPacking8bit* 8 bit packing enabled  
*kSAI\_FifoPacking16bit* 16bit packing enabled

### 38.5.15 enum sai\_sample\_rate\_t

Enumerator

*kSAI\_SampleRate8KHz* Sample rate 8000 Hz.  
*kSAI\_SampleRate11025Hz* Sample rate 11025 Hz.  
*kSAI\_SampleRate12KHz* Sample rate 12000 Hz.  
*kSAI\_SampleRate16KHz* Sample rate 16000 Hz.  
*kSAI\_SampleRate22050Hz* Sample rate 22050 Hz.  
*kSAI\_SampleRate24KHz* Sample rate 24000 Hz.  
*kSAI\_SampleRate32KHz* Sample rate 32000 Hz.  
*kSAI\_SampleRate44100Hz* Sample rate 44100 Hz.  
*kSAI\_SampleRate48KHz* Sample rate 48000 Hz.  
*kSAI\_SampleRate96KHz* Sample rate 96000 Hz.

### 38.5.16 enum sai\_word\_width\_t

Enumerator

*kSAI\_WordWidth8bits* Audio data width 8 bits.  
*kSAI\_WordWidth16bits* Audio data width 16 bits.  
*kSAI\_WordWidth24bits* Audio data width 24 bits.  
*kSAI\_WordWidth32bits* Audio data width 32 bits.

## 38.6 Function Documentation

### 38.6.1 void SAI\_TxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )

Un-gates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_TxGetDefaultConfig\(\)](#).

## Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

### 38.6.2 void SAI\_RxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_RxGetDefaultConfig\(\)](#).

## Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

### 38.6.3 void SAI\_TxGetDefaultConfig ( sai\_config\_t \* *config* )

This API initializes the configuration structure for use in SAI\_TxConfig(). The initialized structure can remain unchanged in SAI\_TxConfig(), or it can be modified before calling SAI\_TxConfig(). This is an example.

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

## Parameters

## Function Documentation

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | pointer to master configuration structure |
|---------------|-------------------------------------------|

### 38.6.4 void SAI\_RxGetDefaultConfig ( sai\_config\_t \* *config* )

This API initializes the configuration structure for use in SAI\_RxConfig(). The initialized structure can remain unchanged in SAI\_RxConfig() or it can be modified before calling SAI\_RxConfig(). This is an example.

```
sai_config_t config;  
SAI_RxGetDefaultConfig(&config);
```

#### Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | pointer to master configuration structure |
|---------------|-------------------------------------------|

### 38.6.5 void SAI\_Deinit ( I2S\_Type \* *base* )

This API gates the SAI clock. The SAI module can't operate unless SAI\_TxInit or SAI\_RxInit is called to enable the clock.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

### 38.6.6 void SAI\_TxReset ( I2S\_Type \* *base* )

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

### 38.6.7 void SAI\_RxReset ( I2S\_Type \* *base* )

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

### 38.6.8 void SAI\_TxEnable ( I2S\_Type \* *base*, bool *enable* )

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SAI base pointer                               |
| <i>enable</i> | True means enable SAI Tx, false means disable. |

### 38.6.9 void SAI\_RxEnable ( I2S\_Type \* *base*, bool *enable* )

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SAI base pointer                               |
| <i>enable</i> | True means enable SAI Rx, false means disable. |

### 38.6.10 static uint32\_t SAI\_TxGetStatusFlag ( I2S\_Type \* *base* ) [inline], [static]

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

### 38.6.11 static void SAI\_TxClearStatusFlags ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

### Parameters

|             |                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                       |
| <i>mask</i> | State mask. It can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_WordStartFlag</li><li>• kSAI_SyncErrorFlag</li><li>• kSAI_FIFOErrorFlag</li></ul> |

**38.6.12** `static uint32_t SAI_RxGetStatusFlag ( I2S_Type * base ) [inline], [static]`

### Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

### Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

**38.6.13** `static void SAI_RxClearStatusFlags ( I2S_Type * base, uint32_t mask ) [inline], [static]`

### Parameters

|             |                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                        |
| <i>mask</i> | State mask. It can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_WordStartFlag</li><li>• kSAI_SyncErrorFlag</li><li>• kSAI_FIFOErrorFlag</li></ul> |

**38.6.14** `void SAI_TxSoftwareReset ( I2S_Type * base, sai_reset_type_t type )`

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.



Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | SAI base pointer                         |
| <i>type</i> | Reset type, FIFO reset or software reset |

### 38.6.15 void SAI\_RxSoftwareReset ( I2S\_Type \* *base*, sai\_reset\_type\_t *type* )

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | SAI base pointer                         |
| <i>type</i> | Reset type, FIFO reset or software reset |

### 38.6.16 void SAI\_TxSetChannelFIFOMask ( I2S\_Type \* *base*, uint8\_t *mask* )

Parameters

|             |                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                 |
| <i>mask</i> | Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled. |

### 38.6.17 void SAI\_RxSetChannelFIFOMask ( I2S\_Type \* *base*, uint8\_t *mask* )

Parameters

|             |                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                 |
| <i>mask</i> | Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled. |

### 38.6.18 void SAI\_TxSetDataOrder ( I2S\_Type \* *base*, sai\_data\_order\_t *order* )

## Function Documentation

Parameters

|              |                       |
|--------------|-----------------------|
| <i>base</i>  | SAI base pointer      |
| <i>order</i> | Data order MSB or LSB |

**38.6.19 void SAI\_RxSetDataOrder ( I2S\_Type \* *base*, sai\_data\_order\_t *order* )**

Parameters

|              |                       |
|--------------|-----------------------|
| <i>base</i>  | SAI base pointer      |
| <i>order</i> | Data order MSB or LSB |

**38.6.20 void SAI\_TxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

|              |                       |
|--------------|-----------------------|
| <i>base</i>  | SAI base pointer      |
| <i>order</i> | Data order MSB or LSB |

**38.6.21 void SAI\_RxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

|              |                       |
|--------------|-----------------------|
| <i>base</i>  | SAI base pointer      |
| <i>order</i> | Data order MSB or LSB |

**38.6.22 void SAI\_TxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

|              |                       |
|--------------|-----------------------|
| <i>base</i>  | SAI base pointer      |
| <i>order</i> | Data order MSB or LSB |

**38.6.23 void SAI\_RxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

|              |                       |
|--------------|-----------------------|
| <i>base</i>  | SAI base pointer      |
| <i>order</i> | Data order MSB or LSB |

**38.6.24 void SAI\_TxSetFIFOPacking ( I2S\_Type \* *base*, sai\_fifo\_packing\_t *pack* )**

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | SAI base pointer.                                    |
| <i>pack</i> | FIFO pack type. It is element of sai_fifo_packing_t. |

**38.6.25 void SAI\_RxSetFIFOPacking ( I2S\_Type \* *base*, sai\_fifo\_packing\_t *pack* )**

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | SAI base pointer.                                    |
| <i>pack</i> | FIFO pack type. It is element of sai_fifo_packing_t. |

**38.6.26 static void SAI\_TxSetFIFOErrorContinue ( I2S\_Type \* *base*, bool *isEnabled* ) [inline], [static]**

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

## Function Documentation

Parameters

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                       |
| <i>isEnabled</i> | Is FIFO error continue enabled, true means enable, false means disable. |

**38.6.27 static void SAI\_RxSetFIFOErrorContinue ( I2S\_Type \* *base*, bool *isEnabled* ) [inline], [static]**

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                       |
| <i>isEnabled</i> | Is FIFO error continue enabled, true means enable, false means disable. |

**38.6.28 static void SAI\_TxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                                                                                                                                                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                             |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFORequestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

**38.6.29 static void SAI\_RxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |                                                                                                                                                                                                                                                                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                   |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**38.6.30 static void SAI\_TxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

## Parameters

|             |                                                                                                                                                                                                                                                                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                   |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

## Function Documentation

### 38.6.31 `static void SAI_RxDisableInterrupts ( I2S_Type * base, uint32_t mask ) [inline], [static]`

#### Parameters

|             |                                                                                                                                                                                                                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                             |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFORequestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

### 38.6.32 `static void SAI_TxEnableDMA ( I2S_Type * base, uint32_t mask, bool enable ) [inline], [static]`

#### Parameters

|               |                                                                                                                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer                                                                                                                                                                                 |
| <i>mask</i>   | DMA source The parameter can be combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_FIFOWarningDMAEnable</li><li>• kSAI_FIFORequestDMAEnable</li></ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.                                                                                                                                                  |

### 38.6.33 `static void SAI_RxEnableDMA ( I2S_Type * base, uint32_t mask, bool enable ) [inline], [static]`

#### Parameters

|               |                                                                                                                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer                                                                                                                                                                                   |
| <i>mask</i>   | DMA source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_FIFOWarningDMAEnable</li><li>• kSAI_FIFORequestDMAEnable</li></ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.                                                                                                                                                    |

**38.6.34** `static uint32_t SAI_TxGetDataRegisterAddress ( I2S_Type * base, uint32_t channel ) [inline], [static]`

This API is used to provide a transfer address for the SAI DMA transfer configuration.

## Function Documentation

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | SAI base pointer.        |
| <i>channel</i> | Which data channel used. |

### Returns

data register address.

**38.6.35** `static uint32_t SAI_RxGetDataRegisterAddress ( I2S_Type * base, uint32_t channel ) [inline], [static]`

This API is used to provide a transfer address for the SAI DMA transfer configuration.

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | SAI base pointer.        |
| <i>channel</i> | Which data channel used. |

### Returns

data register address.

**38.6.36** `void SAI_TxSetFormat ( I2S_Type * base, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

### Parameters

|                           |                                                 |
|---------------------------|-------------------------------------------------|
| <i>base</i>               | SAI base pointer.                               |
| <i>format</i>             | Pointer to the SAI audio data format structure. |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.        |



|                           |                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz. |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|

**38.6.37 void SAI\_RxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

|                           |                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                           |
| <i>format</i>             | Pointer to the SAI audio data format structure.                                                                             |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                    |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz. |

**38.6.38 void SAI\_WriteBlocking ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *bitWidth*, uint8\_t \* *buffer*, uint32\_t *size* )**

Note

This function blocks by polling until data is ready to be sent.

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | SAI base pointer.                                        |
| <i>channel</i>  | Data channel used.                                       |
| <i>bitWidth</i> | How many bits in an audio word; usually 8/16/24/32 bits. |
| <i>buffer</i>   | Pointer to the data to be written.                       |
| <i>size</i>     | Bytes to be written.                                     |

**38.6.39 static void SAI\_WriteData ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *data* ) [inline], [static]**

## Function Documentation

### Parameters

|                |                           |
|----------------|---------------------------|
| <i>base</i>    | SAI base pointer.         |
| <i>channel</i> | Data channel used.        |
| <i>data</i>    | Data needs to be written. |

**38.6.40 void SAI\_ReadBlocking ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *bitWidth*, uint8\_t \* *buffer*, uint32\_t *size* )**

### Note

This function blocks by polling until data is ready to be sent.

### Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | SAI base pointer.                                        |
| <i>channel</i>  | Data channel used.                                       |
| <i>bitWidth</i> | How many bits in an audio word; usually 8/16/24/32 bits. |
| <i>buffer</i>   | Pointer to the data to be read.                          |
| <i>size</i>     | Bytes to be read.                                        |

**38.6.41 static uint32\_t SAI\_ReadData ( I2S\_Type \* *base*, uint32\_t *channel* )  
[inline], [static]**

### Parameters

|                |                    |
|----------------|--------------------|
| <i>base</i>    | SAI base pointer.  |
| <i>channel</i> | Data channel used. |

### Returns

Data in SAI FIFO.

**38.6.42 void SAI\_TransferTxCreateHandle ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

## Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>base</i>     | SAI base pointer                               |
| <i>handle</i>   | SAI handle pointer.                            |
| <i>callback</i> | Pointer to the user callback function.         |
| <i>userData</i> | User parameter passed to the callback function |

### 38.6.43 void SAI\_TransferRxCreateHandle ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

## Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>base</i>     | SAI base pointer.                               |
| <i>handle</i>   | SAI handle pointer.                             |
| <i>callback</i> | Pointer to the user callback function.          |
| <i>userData</i> | User parameter passed to the callback function. |

### 38.6.44 status\_t SAI\_TransferTxSetFormat ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

|                           |                                                 |
|---------------------------|-------------------------------------------------|
| <i>base</i>               | SAI base pointer.                               |
| <i>handle</i>             | SAI handle pointer.                             |
| <i>format</i>             | Pointer to the SAI audio data format structure. |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.        |

## Function Documentation

|                           |                                                                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format. |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------|

Returns

Status of this function. Return value is the status\_t.

**38.6.45** `status_t SAI_TransferRxSetFormat ( I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

|                           |                                                                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                   |
| <i>handle</i>             | SAI handle pointer.                                                                                                                 |
| <i>format</i>             | Pointer to the SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                            |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format. |

Returns

Status of this function. Return value is one of status\_t.

**38.6.46** `status_t SAI_TransferSendNonBlocking ( I2S_Type * base, sai_handle_t * handle, sai_transfer_t * xfer )`

Note

This API returns immediately after the transfer initiates. Call the SAI\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_-SAI\_Busy, the transfer is finished.

## Parameters

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                                   |
| <i>handle</i> | Pointer to the <code>sai_handle_t</code> structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <code>sai_transfer_t</code> structure.                               |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Successfully started the data receive. |
| <i>kStatus_SAI_TxBusy</i>      | Previous receive still not finished.   |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.        |

### 38.6.47 `status_t SAI_TransferReceiveNonBlocking ( I2S_Type * base, sai_handle_t * handle, sai_transfer_t * xfer )`

## Note

This API returns immediately after the transfer initiates. Call the `SAI_RxGetTransferStatusIRQ` to poll the transfer status and check whether the transfer is finished. If the return status is not `kStatus_SAI_Busy`, the transfer is finished.

## Parameters

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer                                                                    |
| <i>handle</i> | Pointer to the <code>sai_handle_t</code> structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <code>sai_transfer_t</code> structure.                               |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Successfully started the data receive. |
| <i>kStatus_SAI_RxBusy</i>      | Previous receive still not finished.   |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.        |

### 38.6.48 `status_t SAI_TransferGetSendCount ( I2S_Type * base, sai_handle_t * handle, size_t * count )`

## Function Documentation

### Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count sent.                                                      |

### Return values

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

### 38.6.49 status\_t SAI\_TransferGetReceiveCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count received.                                                  |

### Return values

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

### 38.6.50 void SAI\_TransferAbortSend ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

### Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |

### 38.6.51 void SAI\_TransferAbortReceive ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

Note

This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer                                                       |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |

### 38.6.52 void SAI\_TransferTerminateSend ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortSend.

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

### 38.6.53 void SAI\_TransferTerminateReceive ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortReceive.

Parameters

---

## Function Documentation

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

### 38.6.54 void SAI\_TransferTxHandleIRQ ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SAI base pointer.                      |
| <i>handle</i> | Pointer to the sai_handle_t structure. |

### 38.6.55 void SAI\_TransferRxHandleIRQ ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SAI base pointer.                      |
| <i>handle</i> | Pointer to the sai_handle_t structure. |



## 38.7 SAI DMA Driver

### 38.7.1 Overview

#### Data Structures

- struct [sai\\_dma\\_handle\\_t](#)  
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

#### Typedefs

- typedef void(\* [sai\\_dma\\_callback\\_t](#))(I2S\_Type \*base, sai\_dma\_handle\_t \*handle, status\_t status, void \*userData)  
*Define SAI DMA callback.*

#### Driver version

- #define [FSL\\_SAI\\_DMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 1, 4))  
*Version 2.1.4.*

#### DMA Transactional

- void [SAI\\_TransferTxCreateHandleDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_dma\\_callback\\_t](#) callback, void \*userData, dma\_handle\_t \*dmaHandle)  
*Initializes the SAI master DMA handle.*
- void [SAI\\_TransferRxCreateHandleDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_dma\\_callback\\_t](#) callback, void \*userData, dma\_handle\_t \*dmaHandle)  
*Initializes the SAI slave DMA handle.*
- void [SAI\\_TransferTxSetFormatDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_TransferRxSetFormatDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- status\_t [SAI\\_TransferSendDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- status\_t [SAI\\_TransferReceiveDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- void [SAI\\_TransferAbortSendDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle)  
*Aborts a SAI transfer using DMA.*
- void [SAI\\_TransferAbortReceiveDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle)  
*Aborts a SAI transfer using DMA.*
- status\_t [SAI\\_TransferGetSendCountDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, size\_t \*count)

## SAI DMA Driver

*Gets byte count sent by SAI.*

- status\_t [SAI\\_TransferGetReceiveCountDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, size\_t \*count)

*Gets byte count received by SAI.*

### 38.7.2 Data Structure Documentation

#### 38.7.2.1 struct \_sai\_dma\_handle

##### Data Fields

- dma\_handle\_t \* [dmaHandle](#)  
*DMA handler for SAI send.*
- uint8\_t [bytesPerFrame](#)  
*Bytes in a frame.*
- uint8\_t [channel](#)  
*Which Data channel SAI use.*
- uint32\_t [state](#)  
*SAI DMA transfer internal state.*
- [sai\\_dma\\_callback\\_t](#) [callback](#)  
*Callback for users while transfer finish or error occurred.*
- void \* [userData](#)  
*User callback parameter.*
- [sai\\_transfer\\_t](#) [saiQueue](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*

##### 38.7.2.1.0.40 Field Documentation

38.7.2.1.0.40.1 [sai\\_transfer\\_t](#) [sai\\_dma\\_handle\\_t::saiQueue](#)[[SAI\\_XFER\\_QUEUE\\_SIZE](#)]

38.7.2.1.0.40.2 [volatile uint8\\_t](#) [sai\\_dma\\_handle\\_t::queueUser](#)

### 38.7.3 Function Documentation

38.7.3.1 [void](#) [SAI\\_TransferTxCreateHandleDMA](#) ( [I2S\\_Type](#) \* *base*, [sai\\_dma\\_handle\\_t](#) \* *handle*, [sai\\_dma\\_callback\\_t](#) *callback*, [void](#) \* *userData*, [dma\\_handle\\_t](#) \* *dmaHandle* )

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

## Parameters

|                  |                                                                     |
|------------------|---------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                   |
| <i>handle</i>    | SAI DMA handle pointer.                                             |
| <i>base</i>      | SAI peripheral base address.                                        |
| <i>callback</i>  | Pointer to user callback function.                                  |
| <i>userData</i>  | User parameter passed to the callback function.                     |
| <i>dmaHandle</i> | DMA handle pointer, this handle shall be static allocated by users. |

**38.7.3.2 void SAI\_TransferRxCreateHandleDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_dma\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *dmaHandle* )**

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

## Parameters

|                  |                                                                     |
|------------------|---------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                   |
| <i>handle</i>    | SAI DMA handle pointer.                                             |
| <i>base</i>      | SAI peripheral base address.                                        |
| <i>callback</i>  | Pointer to user callback function.                                  |
| <i>userData</i>  | User parameter passed to the callback function.                     |
| <i>dmaHandle</i> | DMA handle pointer, this handle shall be static allocated by users. |

**38.7.3.3 void SAI\_TransferTxSetFormatDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *blkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to the format.

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|

## SAI DMA Driver

|                           |                                                                                                                                  |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>             | SAI DMA handle pointer.                                                                                                          |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                      |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                         |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master. clock, this value should equals to masterClockHz in format. |

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully.  |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid. |

**38.7.3.4 void SAI\_TransferRxSetFormatDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets eDMA parameter according to format.

Parameters

|                           |                                                                                                                                  |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                |
| <i>handle</i>             | SAI DMA handle pointer.                                                                                                          |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                      |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                         |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master. clock, this value should equals to masterClockHz in format. |

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully.  |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid. |

**38.7.3.5 status\_t SAI\_TransferSendDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

This interface returns immediately after the transfer initiates. Call the SAI\_GetTransferStatus to poll the transfer status to check whether the SAI transfer finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer.                  |
| <i>handle</i> | SAI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

## Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data receive. |
| <i>kStatus_SAI_TxBusy</i>      | Previous receive still not finished. |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.      |

### 38.7.3.6 status\_t SAI\_TransferReceiveDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This interface returns immediately after transfer initiates. Call SAI\_GetTransferStatus to poll the transfer status to check whether the SAI transfer is finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer                   |
| <i>handle</i> | SAI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

## Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data receive. |
| <i>kStatus_SAI_RxBusy</i>      | Previous receive still not finished. |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.      |

### 38.7.3.7 void SAI\_TransferAbortSendDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle* )

## SAI DMA Driver

### Parameters

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

### 38.7.3.8 void SAI\_TransferAbortReceiveDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle* )

### Parameters

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

### 38.7.3.9 status\_t SAI\_TransferGetSendCountDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI DMA handle pointer.  |
| <i>count</i>  | Bytes count sent by SAI. |

### Return values

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

### 38.7.3.10 status\_t SAI\_TransferGetReceiveCountDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|

|               |                              |
|---------------|------------------------------|
| <i>handle</i> | SAI DMA handle pointer.      |
| <i>count</i>  | Bytes count received by SAI. |

## Return values

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

### 38.8 SAI eDMA Driver

#### 38.8.1 Overview

#### Data Structures

- struct [sai\\_edma\\_handle\\_t](#)  
*SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* [sai\\_edma\\_callback\\_t](#))(I2S\_Type \*base, sai\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*SAI eDMA transfer callback function for finish and error.*

#### Driver version

- #define [FSL\\_SAI\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 1, 4))  
*Version 2.1.4.*

#### eDMA Transactional

- void [SAI\\_TransferTxCreateHandleEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_edma\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*dmaHandle)  
*Initializes the SAI eDMA handle.*
- void [SAI\\_TransferRxCreateHandleEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_edma\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*dmaHandle)  
*Initializes the SAI Rx eDMA handle.*
- void [SAI\\_TransferTxSetFormatEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_TransferRxSetFormatEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- status\_t [SAI\\_TransferSendEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- status\_t [SAI\\_TransferReceiveEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI receive using eDMA.*
- void [SAI\\_TransferTerminateSendEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle)  
*Terminate all SAI send.*
- void [SAI\\_TransferTerminateReceiveEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle)  
*Terminate all SAI receive.*
- void [SAI\\_TransferAbortSendEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle)



- *Aborts a SAI transfer using eDMA.*  
void [SAI\\_TransferAbortReceiveEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle)
- *Aborts a SAI receive using eDMA.*  
status\_t [SAI\\_TransferGetSendCountEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count sent by SAI.*
- status\_t [SAI\\_TransferGetReceiveCountEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count received by SAI.*

## 38.8.2 Data Structure Documentation

### 38.8.2.1 struct \_sai\_edma\_handle

#### Data Fields

- [edma\\_handle\\_t](#) \* [dmaHandle](#)  
*DMA handler for SAI send.*
- uint8\_t [nbytes](#)  
*eDMA minor byte transfer count initially configured.*
- uint8\_t [bytesPerFrame](#)  
*Bytes in a frame.*
- uint8\_t [channel](#)  
*Which data channel.*
- uint8\_t [count](#)  
*The transfer data count in a DMA request.*
- uint32\_t [state](#)  
*Internal state for SAI eDMA transfer.*
- [sai\\_edma\\_callback\\_t](#) [callback](#)  
*Callback for users while transfer finish or error occurs.*
- void \* [userData](#)  
*User callback parameter.*
- uint8\_t [tcd](#) [(SAI\_XFER\_QUEUE\_SIZE+1U)\*sizeof(edma\_tcd\_t)]  
*TCD pool for eDMA transfer.*
- [sai\\_transfer\\_t](#) [saiQueue](#) [SAI\_XFER\_QUEUE\_SIZE]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [SAI\_XFER\_QUEUE\_SIZE]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*

## SAI eDMA Driver

### 38.8.2.1.0.41 Field Documentation

38.8.2.1.0.41.1 `uint8_t sai_edma_handle_t::nbytes`

38.8.2.1.0.41.2 `uint8_t sai_edma_handle_t::tcd[(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`

38.8.2.1.0.41.3 `sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`

38.8.2.1.0.41.4 `volatile uint8_t sai_edma_handle_t::queueUser`

### 38.8.3 Function Documentation

38.8.3.1 `void SAI_TransferTxCreateHandleEDMA ( I2S_Type * base, sai_edma_handle_t * handle, sai_edma_callback_t callback, void * userData, edma_handle_t * dmaHandle )`

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

## Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                    |
| <i>handle</i>    | SAI eDMA handle pointer.                                             |
| <i>base</i>      | SAI peripheral base address.                                         |
| <i>callback</i>  | Pointer to user callback function.                                   |
| <i>userData</i>  | User parameter passed to the callback function.                      |
| <i>dmaHandle</i> | eDMA handle pointer, this handle shall be static allocated by users. |

**38.8.3.2 void SAI\_TransferRxCreateHandleEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_edma\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *dmaHandle* )**

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

## Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                    |
| <i>handle</i>    | SAI eDMA handle pointer.                                             |
| <i>base</i>      | SAI peripheral base address.                                         |
| <i>callback</i>  | Pointer to user callback function.                                   |
| <i>userData</i>  | User parameter passed to the callback function.                      |
| <i>dmaHandle</i> | eDMA handle pointer, this handle shall be static allocated by users. |

**38.8.3.3 void SAI\_TransferTxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *blkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|

## SAI eDMA Driver

|                           |                                                                                                                                 |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>             | SAI eDMA handle pointer.                                                                                                        |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                        |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**38.8.3.4 void SAI\_TransferRxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

### Parameters

|                           |                                                                                                                                      |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                    |
| <i>handle</i>             | SAI eDMA handle pointer.                                                                                                             |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                          |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                             |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format. |

### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**38.8.3.5 status\_t SAI\_TransferSendEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

This interface returns immediately after the transfer initiates. Call SAI\_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SAI base pointer.                      |
| <i>handle</i> | SAI eDMA handle pointer.               |
| <i>xfer</i>   | Pointer to the DMA transfer structure. |

## Return values

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI eDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.      |
| <i>kStatus_TxBusy</i>          | SAI is busy sending data.           |

### 38.8.3.6 status\_t SAI\_TransferReceiveEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This interface returns immediately after the transfer initiates. Call the SAI\_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer                   |
| <i>handle</i> | SAI eDMA handle pointer.           |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI eDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.         |
| <i>kStatus_RxBusy</i>          | SAI is busy receiving data.            |

### 38.8.3.7 void SAI\_TransferTerminateSendEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortSendEDMA.

## SAI eDMA Driver

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

### 38.8.3.8 void SAI\_TransferTerminateReceiveEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle* )

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortReceiveEDMA.

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

### 38.8.3.9 void SAI\_TransferAbortSendEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle* )

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI\_TransferTerminateSendEDMA.

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

### 38.8.3.10 void SAI\_TransferAbortReceiveEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle* )

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI\_TransferTerminateReceiveEDMA.

### Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SAI base pointer |
|-------------|------------------|

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

### 38.8.3.11 **status\_t SAI\_TransferGetSendCountEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |
| <i>count</i>  | Bytes count sent by SAI. |

Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                   |
| <i>kStatus_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |

### 38.8.3.12 **status\_t SAI\_TransferGetReceiveCountEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>handle</i> | SAI eDMA handle pointer.     |
| <i>count</i>  | Bytes count received by SAI. |

Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                   |
| <i>kStatus_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |

### 38.9 SAI SDMA Driver

#### 38.9.1 Overview

#### Data Structures

- struct [sai\\_sdma\\_handle\\_t](#)  
*SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* [sai\\_sdma\\_callback\\_t](#))(I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, status\_t status, void \*userData)  
*SAI SDMA transfer callback function for finish and error.*

#### Driver version

- #define [FSL\\_SAI\\_SDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 1, 4))  
*Version 2.1.4.*

#### SDMA Transactional

- void [SAI\\_TransferTxCreateHandleSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_sdma\\_callback\\_t](#) callback, void \*userData, sdma\_handle\_t \*dmaHandle, uint32\_t eventSource)  
*Initializes the SAI SDMA handle.*
- void [SAI\\_TransferRxCreateHandleSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_sdma\\_callback\\_t](#) callback, void \*userData, sdma\_handle\_t \*dmaHandle, uint32\_t eventSource)  
*Initializes the SAI Rx SDMA handle.*
- void [SAI\\_TransferTxSetFormatSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_TransferRxSetFormatSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- status\_t [SAI\\_TransferSendSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- status\_t [SAI\\_TransferReceiveSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI receive using SDMA.*
- void [SAI\\_TransferAbortSendSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle)  
*Aborts a SAI transfer using SDMA.*
- void [SAI\\_TransferAbortReceiveSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle)  
*Aborts a SAI receive using SDMA.*



## 38.9.2 Data Structure Documentation

### 38.9.2.1 struct \_sai\_sdma\_handle

#### Data Fields

- `sdma_handle_t * dmaHandle`  
*DMA handler for SAI send.*
- `uint8_t bytesPerFrame`  
*Bytes in a frame.*
- `uint8_t channel`  
*Which data channel.*
- `uint8_t count`  
*The transfer data count in a DMA request.*
- `uint32_t state`  
*Internal state for SAI SDMA transfer.*
- `uint32_t eventSource`  
*SAI event source number.*
- `sai_sdma_callback_t callback`  
*Callback for users while transfer finish or error occurs.*
- `void * userData`  
*User callback parameter.*
- `sdma_buffer_descriptor_t bdPool [SAI_XFER_QUEUE_SIZE]`  
*BD pool for SDMA transfer.*
- `sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSize [SAI_XFER_QUEUE_SIZE]`  
*Data bytes need to transfer.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

## SAI SDMA Driver

### 38.9.2.1.0.42 Field Documentation

38.9.2.1.0.42.1 `sdma_buffer_descriptor_t sai_sdma_handle_t::bdPool[SAI_XFER_QUEUE_SIZE]`

38.9.2.1.0.42.2 `sai_transfer_t sai_sdma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`

38.9.2.1.0.42.3 `volatile uint8_t sai_sdma_handle_t::queueUser`

### 38.9.3 Function Documentation

38.9.3.1 `void SAI_TransferTxCreateHandleSDMA ( I2S_Type * base, sai_sdma_handle_t * handle, sai_sdma_callback_t callback, void * userData, sdma_handle_t * dmaHandle, uint32_t eventSource )`

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

## Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                    |
| <i>handle</i>    | SAI SDMA handle pointer.                                             |
| <i>base</i>      | SAI peripheral base address.                                         |
| <i>callback</i>  | Pointer to user callback function.                                   |
| <i>userData</i>  | User parameter passed to the callback function.                      |
| <i>dmaHandle</i> | SDMA handle pointer, this handle shall be static allocated by users. |

**38.9.3.2 void SAI\_TransferRxCreateHandleSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_sdma\_callback\_t *callback*, void \* *userData*, sdma\_handle\_t \* *dmaHandle*, uint32\_t *eventSource* )**

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

## Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                    |
| <i>handle</i>    | SAI SDMA handle pointer.                                             |
| <i>base</i>      | SAI peripheral base address.                                         |
| <i>callback</i>  | Pointer to user callback function.                                   |
| <i>userData</i>  | User parameter passed to the callback function.                      |
| <i>dmaHandle</i> | SDMA handle pointer, this handle shall be static allocated by users. |

**38.9.3.3 void SAI\_TransferTxSetFormatSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *blkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|

## SAI SDMA Driver

|                           |                                                                                                                                 |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>             | SAI SDMA handle pointer.                                                                                                        |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                        |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**38.9.3.4 void SAI\_TransferRxSetFormatSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

Parameters

|                           |                                                                                                                                      |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                    |
| <i>handle</i>             | SAI SDMA handle pointer.                                                                                                             |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                          |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                             |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format. |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**38.9.3.5 status\_t SAI\_TransferSendSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

This interface returns immediately after the transfer initiates. Call SAI\_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SAI base pointer.                      |
| <i>handle</i> | SAI SDMA handle pointer.               |
| <i>xfer</i>   | Pointer to the DMA transfer structure. |

## Return values

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI SDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.      |
| <i>kStatus_TxBusy</i>          | SAI is busy sending data.           |

### 38.9.3.6 status\_t SAI\_TransferReceiveSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This interface returns immediately after the transfer initiates. Call the SAI\_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer                   |
| <i>handle</i> | SAI SDMA handle pointer.           |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI SDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.         |
| <i>kStatus_RxBusy</i>          | SAI is busy receiving data.            |

### 38.9.3.7 void SAI\_TransferAbortSendSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle* )

## SAI SDMA Driver

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI SDMA handle pointer. |

**38.9.3.8 void SAI\_TransferAbortReceiveSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle* )**

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer         |
| <i>handle</i> | SAI SDMA handle pointer. |

## Chapter 39

# SEMC: Smart External DRAM Controller Driver

### 39.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Smart External DRAM Controller block of MCUXpresso SDK devices.

The SEMC Initialize is to initialize for common configure: gate the SEMC clock, configure IOMUX, and queue weight setting. The SEMC Deinitialize is to ungate the clock and disable SEMC module.

The interrupt and disable operation for SEMC.

This group is mainly provide NAND/NOR memory access API which is through IP bus/ IP command access. Since the AXI access is directly read/write is so easy, so the AXI read/write part is not provided in SEMC.

### 39.2 Typical use case

#### Data Structures

- struct [semc\\_sdram\\_config\\_t](#)  
*SEMC SDRAM configuration structure. [More...](#)*
- struct [semc\\_nand\\_timing\\_config\\_t](#)  
*SEMC NAND device timing configuration structure. [More...](#)*
- struct [semc\\_nand\\_config\\_t](#)  
*SEMC NAND configuration structure. [More...](#)*
- struct [semc\\_nor\\_config\\_t](#)  
*SEMC NOR configuration structure. [More...](#)*
- struct [semc\\_sram\\_config\\_t](#)  
*SEMC SRAM configuration structure. [More...](#)*
- struct [semc\\_dbi\\_config\\_t](#)  
*SEMC DBI configuration structure. [More...](#)*
- struct [semc\\_queuea\\_weight\\_t](#)  
*SEMC AXI queue a weight setting. [More...](#)*
- struct [semc\\_queueb\\_weight\\_t](#)  
*SEMC AXI queue b weight setting. [More...](#)*
- struct [semc\\_axi\\_queueweight\\_t](#)  
*SEMC AXI queue weight setting. [More...](#)*
- struct [semc\\_config\\_t](#)  
*SEMC configuration structure. [More...](#)*

#### Enumerations

- enum [\\_semc\\_status](#)  
*SEMC status.*
- enum [semc\\_mem\\_type\\_t](#) {

## Typical use case

- `kSEMC_MemType_SDRAM = 0,`
  - `kSEMC_MemType_SRAM,`
  - `kSEMC_MemType_NOR,`
  - `kSEMC_MemType_NAND,`
  - `kSEMC_MemType_8080 }`
  - SEMC memory device type.*
- enum `semc_waitready_polarity_t` {
  - `kSEMC_LowActive = 0,`
  - `kSEMC_HighActive }`
  - SEMC WAIT/RDY polarity.*
- enum `semc_sdram_cs_t` {
  - `kSEMC_SDRAM_CS0 = 0,`
  - `kSEMC_SDRAM_CS1,`
  - `kSEMC_SDRAM_CS2,`
  - `kSEMC_SDRAM_CS3 }`
  - SEMC SDRAM Chip selection .*
- enum `semc_nand_access_type_t`
  - SEMC NAND device type.*
- enum `semc_interrupt_enable_t` {
  - `kSEMC_IPCmdDoneInterrupt = SEMC_INTEN_IPCMDDONEEN_MASK,`
  - `kSEMC_IPCmdErrInterrupt = SEMC_INTEN_IPCMDERREN_MASK,`
  - `kSEMC_AXICmdErrInterrupt = SEMC_INTEN_AXICMDERREN_MASK,`
  - `kSEMC_AXIBusErrInterrupt = SEMC_INTEN_AXIBUSERREN_MASK }`
  - SEMC interrupts .*
- enum `semc_ipcmd_datasize_t` {
  - `kSEMC_IPcmdDataSize_1bytes = 1,`
  - `kSEMC_IPcmdDataSize_2bytes,`
  - `kSEMC_IPcmdDataSize_3bytes,`
  - `kSEMC_IPcmdDataSize_4bytes }`
  - SEMC IP command data size in bytes.*
- enum `semc_refresh_time_t` {
  - `kSEMC_RefreshThreeClocks = 0x0U,`
  - `kSEMC_RefreshSixClocks,`
  - `kSEMC_RefreshNineClocks }`
  - SEMC auto-refresh timing.*
- enum `semc_caslatency_t` {
  - `kSEMC_LatencyOne = 1,`
  - `kSEMC_LatencyTwo,`
  - `kSEMC_LatencyThree }`
  - CAS latency.*
- enum `semc_sdram_column_bit_num_t` {
  - `kSEMC_SdramColumn_12bit = 0x0U,`
  - `kSEMC_SdramColumn_11bit,`
  - `kSEMC_SdramColumn_10bit,`
  - `kSEMC_SdramColumn_9bit }`
  - SEMC sdram column address bit number.*
- enum `sem_sdram_burst_len_t` {



- ```

kSEMC_Sdram_BurstLen1 = 0,
kSEMC_Sdram_BurstLen2,
kSEMC_Sdram_BurstLen4,
kSEMC_Sdram_BurstLen8 }
    SEMC sdram burst length.

```
- enum `semc_nand_column_bit_num_t` {

```

kSEMC_NandColum_16bit = 0x0U,
kSEMC_NandColum_15bit,
kSEMC_NandColum_14bit,
kSEMC_NandColum_13bit,
kSEMC_NandColum_12bit,
kSEMC_NandColum_11bit,
kSEMC_NandColum_10bit,
kSEMC_NandColum_9bit }
    SEMC nand column address bit number.

```
  - enum `sem_nand_burst_len_t` {

```

kSEMC_Nand_BurstLen1 = 0,
kSEMC_Nand_BurstLen2,
kSEMC_Nand_BurstLen4,
kSEMC_Nand_BurstLen8,
kSEMC_Nand_BurstLen16,
kSEMC_Nand_BurstLen32,
kSEMC_Nand_BurstLen64 }
    SEMC nand burst length.

```
  - enum `semc_norsram_column_bit_num_t` {

```

kSEMC_NorColum_12bit = 0x0U,
kSEMC_NorColum_11bit,
kSEMC_NorColum_10bit,
kSEMC_NorColum_9bit,
kSEMC_NorColum_8bit,
kSEMC_NorColum_7bit,
kSEMC_NorColum_6bit,
kSEMC_NorColum_5bit,
kSEMC_NorColum_4bit,
kSEMC_NorColum_3bit,
kSEMC_NorColum_2bit }
    SEMC nor/sram column address bit number.

```
  - enum `sem_norsram_burst_len_t` {

```

kSEMC_Nor_BurstLen1 = 0,
kSEMC_Nor_BurstLen2,
kSEMC_Nor_BurstLen4,
kSEMC_Nor_BurstLen8,
kSEMC_Nor_BurstLen16,
kSEMC_Nor_BurstLen32,
kSEMC_Nor_BurstLen64 }
    SEMC nor/sram burst length.

```

## Typical use case

- enum `semc_dbi_column_bit_num_t` {  
    `kSEMC_Dbi_Colum_12bit` = 0x0U,  
    `kSEMC_Dbi_Colum_11bit`,  
    `kSEMC_Dbi_Colum_10bit`,  
    `kSEMC_Dbi_Colum_9bit`,  
    `kSEMC_Dbi_Colum_8bit`,  
    `kSEMC_Dbi_Colum_7bit`,  
    `kSEMC_Dbi_Colum_6bit`,  
    `kSEMC_Dbi_Colum_5bit`,  
    `kSEMC_Dbi_Colum_4bit`,  
    `kSEMC_Dbi_Colum_3bit`,  
    `kSEMC_Dbi_Colum_2bit` }  
    *SEMC dbi column address bit number.*
- enum `sem_dbi_burst_len_t` {  
    `kSEMC_Dbi_BurstLen1` = 0,  
    `kSEMC_Dbi_BurstLen2`,  
    `kSEMC_Dbi_Dbi_BurstLen4`,  
    `kSEMC_Dbi_BurstLen8`,  
    `kSEMC_Dbi_BurstLen16`,  
    `kSEMC_Dbi_BurstLen32`,  
    `kSEMC_Dbi_BurstLen64` }  
    *SEMC dbi burst length.*
- enum `semc_iomux_pin` {  
    `kSEMC_MUXA8` = `SEMC_IOCRR_MUX_A8_SHIFT`,  
    `kSEMC_MUXCSX0` = `SEMC_IOCRR_MUX_CSX0_SHIFT`,  
    `kSEMC_MUXCSX1` = `SEMC_IOCRR_MUX_CSX1_SHIFT`,  
    `kSEMC_MUXCSX2` = `SEMC_IOCRR_MUX_CSX2_SHIFT`,  
    `kSEMC_MUXCSX3` = `SEMC_IOCRR_MUX_CSX3_SHIFT`,  
    `kSEMC_MUXRDY` = `SEMC_IOCRR_MUX_RDY_SHIFT` }  
    *SEMC IOMUXC.*
- enum `semc_iomux_nora27_pin` {  
    `kSEMC_MORA27_NONE` = 0,  
    `kSEMC_NORA27_MUXCSX3` = `SEMC_IOCRR_MUX_CSX3_SHIFT`,  
    `kSEMC_NORA27_MUXRDY` = `SEMC_IOCRR_MUX_RDY_SHIFT` }  
    *SEMC NOR/PSRAM Address bit 27 A27.*
- enum `smec_port_size_t` {  
    `kSEMC_PortSize8Bit` = 0,  
    `kSEMC_PortSize16Bit` }  
    *SEMC port size.*
- enum `semc_addr_mode_t` {  
    `kSEMC_AddrDataMux` = 0,  
    `kSEMC_AdvAddrdataMux`,  
    `kSEMC_AddrDataNonMux` }  
    *SEMC address mode.*
- enum `semc_dqs_mode_t` {  
    `kSEMC_Loopbackinternal` = 0,

- ```
kSEMC_Loopbackdqspad }
```
- *SEMC DQS read strobe mode.*
  - enum `semc_adv_polarity_t` {  
`kSEMC_AdvActiveLow` = 0,  
`kSEMC_AdvActivehigh` }
  - *SEMC ADV signal active polarity.*
  - enum `semc_rdy_polarity_t` {  
`kSEMC_RdyActiveLow` = 0,  
`kSEMC_RdyActivehigh` }
  - *SEMC RDY signal active polarity.*
  - enum `semc_ipcmd_nand_addrmode_t` {  
`kSEMC_NANDAM_ColumnRow` = 0x0U,  
`kSEMC_NANDAM_ColumnCA0`,  
`kSEMC_NANDAM_ColumnCA0CA1`,  
`kSEMC_NANDAM_RawRA0`,  
`kSEMC_NANDAM_RawRA0RA1`,  
`kSEMC_NANDAM_RawRA0RA1RA2` }
  - *SEMC IP command for NAND: address mode.*
  - enum `semc_ipcmd_nand_cmdmode_t` {  
`kSEMC_NANDCM_Command` = 0x2U,  
`kSEMC_NANDCM_CommandHold`,  
`kSEMC_NANDCM_CommandAddress`,  
`kSEMC_NANDCM_CommandAddressHold`,  
`kSEMC_NANDCM_CommandAddressRead`,  
`kSEMC_NANDCM_CommandAddressWrite`,  
`kSEMC_NANDCM_CommandRead`,  
`kSEMC_NANDCM_CommandWrite`,  
`kSEMC_NANDCM_Read`,  
`kSEMC_NANDCM_Write` }
  - *SEMC IP command for NAND command mode.*
  - enum `semc_nand_address_option_t` {  
`kSEMC_NandAddrOption_5byte_CA2RA3` = 0U,  
`kSEMC_NandAddrOption_4byte_CA2RA2` = 2U,  
`kSEMC_NandAddrOption_3byte_CA2RA1` = 4U,  
`kSEMC_NandAddrOption_4byte_CA1RA3` = 1U,  
`kSEMC_NandAddrOption_3byte_CA1RA2` = 3U,  
`kSEMC_NandAddrOption_2byte_CA1RA1` = 7U }
  - *SEMC NAND address option.*
  - enum `semc_ipcmd_nor_dbi_t` {  
`kSEMC_NORDBICM_Read` = 0x2U,  
`kSEMC_NORDBICM_Write` }
  - *SEMC IP command for NOR.*
  - enum `semc_ipcmd_sram_t` {  
`kSEMC_SRAMCM_ArrayRead` = 0x2U,  
`kSEMC_SRAMCM_ArrayWrite`,  
`kSEMC_SRAMCM_RegRead`,

## Typical use case

- `kSEMC_SRAMCM_RegWrite` }  
*SEMC IP command for SRAM.*
- enum `semc_ipcmd_sdram_t` {  
`kSEMC_SDRAMCM_Read` = 0x8U,  
`kSEMC_SDRAMCM_Write`,  
`kSEMC_SDRAMCM_Modeset`,  
`kSEMC_SDRAMCM_Active`,  
`kSEMC_SDRAMCM_AutoRefresh`,  
`kSEMC_SDRAMCM_SelfRefresh`,  
`kSEMC_SDRAMCM_Precharge`,  
`kSEMC_SDRAMCM_Prechargeall` }  
*SEMC IP command for SDARM.*

## Driver version

- #define `FSL_SEMC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*SEMC driver version 2.0.1.*

## SEMC Initialization and De-initialization

- void `SEMC_GetDefaultConfig` (`semc_config_t` \*config)  
*Gets the SEMC default basic configuration structure.*
- void `SEMC_Init` (`SEMC_Type` \*base, `semc_config_t` \*configure)  
*Initializes SEMC.*
- void `SEMC_Deinit` (`SEMC_Type` \*base)  
*Deinitializes the SEMC module and gates the clock.*

## SEMC Configuration Operation For Each Memory Type

- status\_t `SEMC_ConfigureSDRAM` (`SEMC_Type` \*base, `semc_sdram_cs_t` cs, `semc_sdram_config_t` \*config, uint32\_t clkSrc\_Hz)  
*Configures SDRAM controller in SEMC.*
- status\_t `SEMC_ConfigureNAND` (`SEMC_Type` \*base, `semc_nand_config_t` \*config, uint32\_t clkSrc\_Hz)  
*Configures NAND controller in SEMC.*
- status\_t `SEMC_ConfigureNOR` (`SEMC_Type` \*base, `semc_nor_config_t` \*config, uint32\_t clkSrc\_Hz)  
*Configures NOR controller in SEMC.*
- status\_t `SEMC_ConfigureSRAM` (`SEMC_Type` \*base, `semc_sram_config_t` \*config, uint32\_t clkSrc\_Hz)  
*Configures SRAM controller in SEMC.*
- status\_t `SEMC_ConfigureDBI` (`SEMC_Type` \*base, `semc_dbi_config_t` \*config, uint32\_t clkSrc\_Hz)  
*Configures DBI controller in SEMC.*

## SEMC Interrupt Operation

- static void `SEMC_EnableInterrupts` (`SEMC_Type` \*base, uint32\_t mask)  
*Enables the SEMC interrupt.*

- static void [SEMC\\_DisableInterrupts](#) (SEMC\_Type \*base, uint32\_t mask)  
*Disables the SEMC interrupt.*
- static bool [SEMC\\_GetStatusFlag](#) (SEMC\_Type \*base)  
*Gets the SEMC status.*
- static void [SEMC\\_ClearStatusFlags](#) (SEMC\_Type \*base, uint32\_t mask)  
*Clears the SEMC status flag state.*

## SEMC Memory Access Operation

- static bool [SEMC\\_IsInIdle](#) (SEMC\_Type \*base)  
*Check if SEMC is in idle.*
- status\_t [SEMC\\_SendIPCommand](#) (SEMC\_Type \*base, [semc\\_mem\\_type\\_t](#) type, uint32\_t address, uint16\_t command, uint32\_t write, uint32\_t \*read)  
*SEMC IP command access.*
- static uint16\_t [SEMC\\_BuildNandIPCommand](#) (uint8\_t userCommand, [semc\\_ipcmd\\_nand\\_addrmode\\_t](#) addrMode, [semc\\_ipcmd\\_nand\\_cmdmode\\_t](#) cmdMode)  
*Build SEMC IP command for NAND.*
- static bool [SEMC\\_IsNandReady](#) (SEMC\_Type \*base)  
*Check if the NAND device is ready.*
- status\_t [SEMC\\_IPCommandNandWrite](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NAND device memory write through IP command.*
- status\_t [SEMC\\_IPCommandNandRead](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NAND device memory read through IP command.*
- status\_t [SEMC\\_IPCommandNorWrite](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NOR device memory write through IP command.*
- status\_t [SEMC\\_IPCommandNorRead](#) (SEMC\_Type \*base, uint32\_t address, uint8\_t \*data, uint32\_t size\_bytes)  
*SEMC NOR device memory read through IP command.*

## 39.3 Data Structure Documentation

### 39.3.1 struct [semc\\_sdram\\_config\\_t](#)

1. The memory size in the configuration is in the unit of KB. So memsize\_kbytes should be set as  $2^2$ ,  $2^3$ ,  $2^4$  .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.
2. The prescalePeriod\_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer\_n16cycle = 0, it means the prescaler timer period is  $256 * 16$  clock cycles. For precalerIf precalerTimer\_n16cycle not equal to 0, The prescaler timer period is prescalePeriod\_N16Cycle \* 16 clock cycles. idleTimeout\_NprescalePeriod, refreshUrgThreshold\_NprescalePeriod, refreshPeriod\_NprescalePeriod are similar to prescalePeriod\_N16Cycle.

## Data Fields

- [semc\\_iomux\\_pin csxPinMux](#)

## Data Structure Documentation

- *CS pin mux.*  
• `uint32_t address`  
*The base address.*
- `uint32_t memsize_kbytes`  
*The memory size in unit of kbytes.*
- `smec_port_size_t portSize`  
*Port size.*
- `sem_sdram_burst_len_t burstLen`  
*Burst length.*
- `semc_sdram_column_bit_num_t columnAddrBitNum`  
*Column address bit number.*
- `semc_caslatency_t casLatency`  
*CAS latency.*
- `uint8_t tPrecharge2Act_Ns`  
*Precharge to active wait time in unit of nanosecond.*
- `uint8_t tAct2ReadWrite_Ns`  
*Act to read/write wait time in unit of nanosecond.*
- `uint8_t tRefreshRecovery_Ns`  
*Refresh recovery time in unit of nanosecond.*
- `uint8_t tWriteRecovery_Ns`  
*write recovery time in unit of nanosecond.*
- `uint8_t tCkeOff_Ns`  
*CKE off minimum time in unit of nanosecond.*
- `uint8_t tAct2Prechage_Ns`  
*Active to precharge in unit of nanosecond.*
- `uint8_t tSelfRefRecovery_Ns`  
*Self refresh recovery time in unit of nanosecond.*
- `uint8_t tRefresh2Refresh_Ns`  
*Refresh to refresh wait time in unit of nanosecond.*
- `uint8_t tAct2Act_Ns`  
*Active to active wait time in unit of nanosecond.*
- `uint32_t tPrescalePeriod_Ns`  
*Prescaler timer period should not be larger than  $256 * 16 * \text{clock cycle}$ .*
- `uint32_t tIdleTimeout_Ns`  
*Idle timeout in unit of prescale time period.*
- `uint32_t refreshPeriod_nsPerRow`  
*Refresh timer period like  $64\text{ms} * 1000000/8192$ .*
- `uint32_t refreshUrgThreshold`  
*Refresh urgent threshold.*
- `uint8_t refreshBurstLen`  
*Refresh burst length.*

### 39.3.1.0.0.43 Field Documentation

#### 39.3.1.0.0.43.1 `semc_iomux_pin semc_sdram_config_t::csxPinMux`

The `kSEMC_MUXA8` is not valid in sdram pin mux setting.

- 39.3.1.0.0.43.2 `uint32_t semc_sdram_config_t::address`
  - 39.3.1.0.0.43.3 `uint32_t semc_sdram_config_t::memsize_kbytes`
  - 39.3.1.0.0.43.4 `smec_port_size_t semc_sdram_config_t::portSize`
  - 39.3.1.0.0.43.5 `sem_sdram_burst_len_t semc_sdram_config_t::burstLen`
  - 39.3.1.0.0.43.6 `semc_sdram_column_bit_num_t semc_sdram_config_t::columnAddrBitNum`
  - 39.3.1.0.0.43.7 `semc_caslatency_t semc_sdram_config_t::casLatency`
  - 39.3.1.0.0.43.8 `uint8_t semc_sdram_config_t::tPrecharge2Act_Ns`
  - 39.3.1.0.0.43.9 `uint8_t semc_sdram_config_t::tAct2ReadWrite_Ns`
  - 39.3.1.0.0.43.10 `uint8_t semc_sdram_config_t::tRefreshRecovery_Ns`
  - 39.3.1.0.0.43.11 `uint8_t semc_sdram_config_t::tWriteRecovery_Ns`
  - 39.3.1.0.0.43.12 `uint8_t semc_sdram_config_t::tCkeOff_Ns`
  - 39.3.1.0.0.43.13 `uint8_t semc_sdram_config_t::tAct2Prechage_Ns`
  - 39.3.1.0.0.43.14 `uint8_t semc_sdram_config_t::tSelfRefRecovery_Ns`
  - 39.3.1.0.0.43.15 `uint8_t semc_sdram_config_t::tRefresh2Refresh_Ns`
  - 39.3.1.0.0.43.16 `uint8_t semc_sdram_config_t::tAct2Act_Ns`
  - 39.3.1.0.0.43.17 `uint32_t semc_sdram_config_t::tPrescalePeriod_Ns`
  - 39.3.1.0.0.43.18 `uint32_t semc_sdram_config_t::tIdleTimeout_Ns`
  - 39.3.1.0.0.43.19 `uint32_t semc_sdram_config_t::refreshPeriod_nsPerRow`
  - 39.3.1.0.0.43.20 `uint32_t semc_sdram_config_t::refreshUrgThreshold`
  - 39.3.1.0.0.43.21 `uint8_t semc_sdram_config_t::refreshBurstLen`
- 39.3.2 struct semc\_nand\_timing\_config\_t**

### Data Fields

- `uint8_t tCeSetup_Ns`  
*CE setup time: tCS.*
- `uint8_t tCeHold_Ns`  
*CE hold time: tCH.*
- `uint8_t tCeInterval_Ns`  
*CE interval time:tCEITV.*

## Data Structure Documentation

- uint8\_t [tWeLow\\_Ns](#)  
*WE low time: tWP.*
- uint8\_t [tWeHigh\\_Ns](#)  
*WE high time: tWH.*
- uint8\_t [tReLow\\_Ns](#)  
*RE low time: tRP.*
- uint8\_t [tReHigh\\_Ns](#)  
*RE high time: tREH.*
- uint8\_t [tTurnAround\\_Ns](#)  
*Turnaround time for async mode: tTA.*
- uint8\_t [tWehigh2Relow\\_Ns](#)  
*WE# high to RE# wait time: tWHR.*
- uint8\_t [tRehigh2Welow\\_Ns](#)  
*RE# high to WE# low wait time: tRHW.*
- uint8\_t [tAle2WriteStart\\_Ns](#)  
*ALE to write start wait time: tADL.*
- uint8\_t [tReady2Relow\\_Ns](#)  
*Ready to RE# low min wait time: tRR.*
- uint8\_t [tWehigh2Busy\\_Ns](#)  
*WE# high to busy wait time: tWB.*



**39.3.2.0.0.44 Field Documentation**

- 39.3.2.0.0.44.1** `uint8_t semc_nand_timing_config_t::tCeSetup_Ns`
- 39.3.2.0.0.44.2** `uint8_t semc_nand_timing_config_t::tCeHold_Ns`
- 39.3.2.0.0.44.3** `uint8_t semc_nand_timing_config_t::tCeInterval_Ns`
- 39.3.2.0.0.44.4** `uint8_t semc_nand_timing_config_t::tWeLow_Ns`
- 39.3.2.0.0.44.5** `uint8_t semc_nand_timing_config_t::tWeHigh_Ns`
- 39.3.2.0.0.44.6** `uint8_t semc_nand_timing_config_t::tReLow_Ns`
- 39.3.2.0.0.44.7** `uint8_t semc_nand_timing_config_t::tReHigh_Ns`
- 39.3.2.0.0.44.8** `uint8_t semc_nand_timing_config_t::tTurnAround_Ns`
- 39.3.2.0.0.44.9** `uint8_t semc_nand_timing_config_t::tWehigh2Relow_Ns`
- 39.3.2.0.0.44.10** `uint8_t semc_nand_timing_config_t::tRehigh2Welow_Ns`
- 39.3.2.0.0.44.11** `uint8_t semc_nand_timing_config_t::tAle2WriteStart_Ns`
- 39.3.2.0.0.44.12** `uint8_t semc_nand_timing_config_t::tReady2Relow_Ns`
- 39.3.2.0.0.44.13** `uint8_t semc_nand_timing_config_t::tWehigh2Busy_Ns`

**39.3.3 struct semc\_nand\_config\_t****Data Fields**

- [semc\\_iomux\\_pin cePinMux](#)  
*The CE pin mux setting.*
- `uint32_t axiAddress`  
*The base address for AXI nand.*
- `uint32_t axiMemsize_kbytes`  
*The memory size in unit of kbytes for AXI nand.*
- `uint32_t ipgAddress`  
*The base address for IPG nand .*
- `uint32_t ipgMemsize_kbytes`  
*The memory size in unit of kbytes for IPG nand.*
- [semc\\_rdy\\_polarity\\_t rdyactivePolarity](#)  
*Wait ready polarity.*
- `bool edoModeEnabled`  
*EDO mode enabled.*
- [semc\\_nand\\_column\\_bit\\_num\\_t columnAddrBitNum](#)  
*Column address bit number.*
- [semc\\_nand\\_address\\_option\\_t arrayAddrOption](#)  
*Address option.*

## Data Structure Documentation

- [sem\\_nand\\_burst\\_len\\_t burstLen](#)  
*Burst length.*
- [smec\\_port\\_size\\_t portSize](#)  
*Port size.*
- [semc\\_nand\\_timing\\_config\\_t \\* timingConfig](#)  
*SEMC nand timing configuration.*

### 39.3.3.0.0.45 Field Documentation

#### 39.3.3.0.0.45.1 semc\_iomux\_pin semc\_nand\_config\_t::cePinMux

The kSEMC\_MUXRDY is not valid for CE pin setting.

#### 39.3.3.0.0.45.2 uint32\_t semc\_nand\_config\_t::axiAddress

#### 39.3.3.0.0.45.3 uint32\_t semc\_nand\_config\_t::axiMemsize\_kbytes

#### 39.3.3.0.0.45.4 uint32\_t semc\_nand\_config\_t::ipgAddress

#### 39.3.3.0.0.45.5 uint32\_t semc\_nand\_config\_t::ipgMemsize\_kbytes

#### 39.3.3.0.0.45.6 semc\_rdy\_polarity\_t semc\_nand\_config\_t::rdyactivePolarity

#### 39.3.3.0.0.45.7 bool semc\_nand\_config\_t::edoModeEnabled

#### 39.3.3.0.0.45.8 semc\_nand\_column\_bit\_num\_t semc\_nand\_config\_t::columnAddrBitNum

#### 39.3.3.0.0.45.9 semc\_nand\_address\_option\_t semc\_nand\_config\_t::arrayAddrOption

#### 39.3.3.0.0.45.10 sem\_nand\_burst\_len\_t semc\_nand\_config\_t::burstLen

#### 39.3.3.0.0.45.11 smec\_port\_size\_t semc\_nand\_config\_t::portSize

#### 39.3.3.0.0.45.12 semc\_nand\_timing\_config\_t\* semc\_nand\_config\_t::timingConfig

### 39.3.4 struct semc\_nor\_config\_t

#### Data Fields

- [semc\\_iomux\\_pin cePinMux](#)  
*The CE# pin mux setting.*
- [semc\\_iomux\\_nora27\\_pin addr27](#)  
*The Addr bit 27 pin mux setting.*
- [uint32\\_t address](#)  
*The base address.*
- [uint32\\_t memsize\\_kbytes](#)  
*The memory size in unit of kbytes.*
- [uint8\\_t addrPortWidth](#)  
*The address port width.*
- [semc\\_rdy\\_polarity\\_t rdyactivePolarity](#)

- *Wait ready polarity.*
- [semc\\_adv\\_polarity\\_t advActivePolarity](#)  
*ADV# polarity.*
- [semc\\_norsram\\_column\\_bit\\_num\\_t columnAddrBitNum](#)  
*Column address bit number.*
- [semc\\_addr\\_mode\\_t addrMode](#)  
*Address mode.*
- [sem\\_norsram\\_burst\\_len\\_t burstLen](#)  
*Burst length.*
- [smec\\_port\\_size\\_t portSize](#)  
*Port size.*
- [uint8\\_t tCeSetup\\_Ns](#)  
*The CE setup time.*
- [uint8\\_t tCeHold\\_Ns](#)  
*The CE hold time.*
- [uint8\\_t tCeInterval\\_Ns](#)  
*CE interval minimum time.*
- [uint8\\_t tAddrSetup\\_Ns](#)  
*The address setup time.*
- [uint8\\_t tAddrHold\\_Ns](#)  
*The address hold time.*
- [uint8\\_t tWeLow\\_Ns](#)  
*WE low time for async mode.*
- [uint8\\_t tWeHigh\\_Ns](#)  
*WE high time for async mode.*
- [uint8\\_t tReLow\\_Ns](#)  
*RE low time for async mode.*
- [uint8\\_t tReHigh\\_Ns](#)  
*RE high time for async mode.*
- [uint8\\_t tTurnAround\\_Ns](#)  
*Turnaround time for async mode.*
- [uint8\\_t tAddr2WriteHold\\_Ns](#)  
*Address to write data hold time for async mode.*
- [uint8\\_t tWriteSetup\\_Ns](#)  
*Write data setup time for sync mode.*
- [uint8\\_t tWriteHold\\_Ns](#)  
*Write hold time for sync mode.*
- [uint8\\_t latencyCount](#)  
*Latency count for sync mode.*
- [uint8\\_t readCycle](#)  
*Read cycle time for sync mode.*



## 39.3.4.0.0.46 Field Documentation

- 39.3.4.0.0.46.1 `semc_iomux_pin` `semc_nor_config_t::cePinMux`
- 39.3.4.0.0.46.2 `semc_iomux_nora27_pin` `semc_nor_config_t::addr27`
- 39.3.4.0.0.46.3 `uint32_t` `semc_nor_config_t::address`
- 39.3.4.0.0.46.4 `uint32_t` `semc_nor_config_t::memsize_kbytes`
- 39.3.4.0.0.46.5 `uint8_t` `semc_nor_config_t::addrPortWidth`
- 39.3.4.0.0.46.6 `semc_rdy_polarity_t` `semc_nor_config_t::rdyactivePolarity`
- 39.3.4.0.0.46.7 `semc_adv_polarity_t` `semc_nor_config_t::advActivePolarity`
- 39.3.4.0.0.46.8 `semc_norsram_column_bit_num_t` `semc_nor_config_t::columnAddrBitNum`
- 39.3.4.0.0.46.9 `semc_addr_mode_t` `semc_nor_config_t::addrMode`
- 39.3.4.0.0.46.10 `sem_norsram_burst_len_t` `semc_nor_config_t::burstLen`
- 39.3.4.0.0.46.11 `smec_port_size_t` `semc_nor_config_t::portSize`
- 39.3.4.0.0.46.12 `uint8_t` `semc_nor_config_t::tCeSetup_Ns`
- 39.3.4.0.0.46.13 `uint8_t` `semc_nor_config_t::tCeHold_Ns`
- 39.3.4.0.0.46.14 `uint8_t` `semc_nor_config_t::tCeInterval_Ns`
- 39.3.4.0.0.46.15 `uint8_t` `semc_nor_config_t::tAddrSetup_Ns`
- 39.3.4.0.0.46.16 `uint8_t` `semc_nor_config_t::tAddrHold_Ns`
- 39.3.4.0.0.46.17 `uint8_t` `semc_nor_config_t::tWeLow_Ns`
- 39.3.4.0.0.46.18 `uint8_t` `semc_nor_config_t::tWeHigh_Ns`
- 39.3.4.0.0.46.19 `uint8_t` `semc_nor_config_t::tReLow_Ns`
- 39.3.4.0.0.46.20 `uint8_t` `semc_nor_config_t::tReHigh_Ns`
- 39.3.4.0.0.46.21 `uint8_t` `semc_nor_config_t::tTurnAround_Ns`
- 39.3.4.0.0.46.22 `uint8_t` `semc_nor_config_t::tAddr2WriteHold_Ns`
- 39.3.4.0.0.46.23 `uint8_t` `semc_nor_config_t::tWriteSetup_Ns`
- 39.3.4.0.0.46.24 `uint8_t` `semc_nor_config_t::tWriteHold_Ns`
- 39.3.4.0.0.46.25 `uint8_t` `semc_nor_config_t::latencyCount`
- 39.3.4.0.0.46.26 `uint8_t` `semc_nor_config_t::readCycle`

## Data Structure Documentation

- *The CE# pin mux setting.*  
• [semc\\_iomux\\_nora27\\_pin addr27](#)  
*The Addr bit 27 pin mux setting.*
- [uint32\\_t address](#)  
*The base address.*
- [uint32\\_t memsize\\_kbytes](#)  
*The memory size in unit of kbytes.*
- [uint8\\_t addrPortWidth](#)  
*The address port width.*
- [semc\\_adv\\_polarity\\_t advActivePolarity](#)  
*ADV# polarity 1: active high, 0: active low.*
- [semc\\_addr\\_mode\\_t addrMode](#)  
*Address mode.*
- [sem\\_norsram\\_burst\\_len\\_t burstLen](#)  
*Burst length.*
- [smec\\_port\\_size\\_t portSize](#)  
*Port size.*
- [uint8\\_t tCeSetup\\_Ns](#)  
*The CE setup time.*
- [uint8\\_t tCeHold\\_Ns](#)  
*The CE hold time.*
- [uint8\\_t tCeInterval\\_Ns](#)  
*CE interval minimum time.*
- [uint8\\_t tAddrSetup\\_Ns](#)  
*The address setup time.*
- [uint8\\_t tAddrHold\\_Ns](#)  
*The address hold time.*
- [uint8\\_t tWeLow\\_Ns](#)  
*WE low time for async mode.*
- [uint8\\_t tWeHigh\\_Ns](#)  
*WE high time for async mode.*
- [uint8\\_t tReLow\\_Ns](#)  
*RE low time for async mode.*
- [uint8\\_t tReHigh\\_Ns](#)  
*RE high time for async mode.*
- [uint8\\_t tTurnAround\\_Ns](#)  
*Turnaround time for async mode.*
- [uint8\\_t tAddr2WriteHold\\_Ns](#)  
*Address to write data hold time for async mode.*
- [uint8\\_t tWriteSetup\\_Ns](#)  
*Write data setup time for sync mode.*
- [uint8\\_t tWriteHold\\_Ns](#)  
*Write hold time for sync mode.*
- [uint8\\_t latencyCount](#)  
*Latency count for sync mode.*
- [uint8\\_t readCycle](#)  
*Read cycle time for sync mode.*



## Data Structure Documentation

### 39.3.5.0.0.47 Field Documentation

- 39.3.5.0.0.47.1 `semc_iomux_pin semc_sram_config_t::cePinMux`
- 39.3.5.0.0.47.2 `semc_iomux_nora27_pin semc_sram_config_t::addr27`
- 39.3.5.0.0.47.3 `uint32_t semc_sram_config_t::address`
- 39.3.5.0.0.47.4 `uint32_t semc_sram_config_t::memsize_kbytes`
- 39.3.5.0.0.47.5 `uint8_t semc_sram_config_t::addrPortWidth`
- 39.3.5.0.0.47.6 `semc_adv_polarity_t semc_sram_config_t::advActivePolarity`
- 39.3.5.0.0.47.7 `semc_addr_mode_t semc_sram_config_t::addrMode`
- 39.3.5.0.0.47.8 `sem_norsram_burst_len_t semc_sram_config_t::burstLen`
- 39.3.5.0.0.47.9 `smec_port_size_t semc_sram_config_t::portSize`
- 39.3.5.0.0.47.10 `uint8_t semc_sram_config_t::tCeSetup_Ns`
- 39.3.5.0.0.47.11 `uint8_t semc_sram_config_t::tCeHold_Ns`
- 39.3.5.0.0.47.12 `uint8_t semc_sram_config_t::tCeInterval_Ns`
- 39.3.5.0.0.47.13 `uint8_t semc_sram_config_t::tAddrSetup_Ns`
- 39.3.5.0.0.47.14 `uint8_t semc_sram_config_t::tAddrHold_Ns`
- 39.3.5.0.0.47.15 `uint8_t semc_sram_config_t::tWeLow_Ns`
- 39.3.5.0.0.47.16 `uint8_t semc_sram_config_t::tWeHigh_Ns`
- 39.3.5.0.0.47.17 `uint8_t semc_sram_config_t::tReLow_Ns`
- 39.3.5.0.0.47.18 `uint8_t semc_sram_config_t::tReHigh_Ns`
- 39.3.5.0.0.47.19 `uint8_t semc_sram_config_t::tTurnAround_Ns`
- 39.3.5.0.0.47.20 `uint8_t semc_sram_config_t::tAddr2WriteHold_Ns`
- 39.3.5.0.0.47.21 `uint8_t semc_sram_config_t::tWriteSetup_Ns`
- 39.3.5.0.0.47.22 `uint8_t semc_sram_config_t::tWriteHold_Ns`
- 39.3.5.0.0.47.23 `uint8_t semc_sram_config_t::latencyCount`
- 39.3.5.0.0.47.24 `uint8_t semc_sram_config_t::readCycle`

### 39.3.6 `struct semc_dbi_config_t`

#### Data Fields



- *The CE# pin mux.*  
uint32\_t address
- *The base address.*  
uint32\_t memsize\_kbytes
- *The memory size in unit of 4kbytes.*  
semc\_dbi\_column\_bit\_num\_t columnAddrBitNum
- *Column address bit number.*  
sem\_dbi\_burst\_len\_t burstLen
- *Burst length.*  
smec\_port\_size\_t portSize
- *Port size.*  
uint8\_t tCsxSetup\_Ns
- *The CSX setup time.*  
uint8\_t tCsxHold\_Ns
- *The CSX hold time.*  
uint8\_t tWexLow\_Ns
- *WEX low time.*  
uint8\_t tWexHigh\_Ns
- *WEX high time.*  
uint8\_t tRdxLow\_Ns
- *RDX low time.*  
uint8\_t tRdxHigh\_Ns
- *RDX high time.*  
uint8\_t tCsxInterval\_Ns
- *Write data setup time.*

## Data Structure Documentation

### 39.3.6.0.0.48 Field Documentation

39.3.6.0.0.48.1 `semc_iomux_pin semc_dbi_config_t::csxPinMux`

39.3.6.0.0.48.2 `uint32_t semc_dbi_config_t::address`

39.3.6.0.0.48.3 `uint32_t semc_dbi_config_t::memsize_kbytes`

39.3.6.0.0.48.4 `semc_dbi_column_bit_num_t semc_dbi_config_t::columnAddrBitNum`

39.3.6.0.0.48.5 `sem_dbi_burst_len_t semc_dbi_config_t::burstLen`

39.3.6.0.0.48.6 `smec_port_size_t semc_dbi_config_t::portSize`

39.3.6.0.0.48.7 `uint8_t semc_dbi_config_t::tCsxSetup_Ns`

39.3.6.0.0.48.8 `uint8_t semc_dbi_config_t::tCsxHold_Ns`

39.3.6.0.0.48.9 `uint8_t semc_dbi_config_t::tWexLow_Ns`

39.3.6.0.0.48.10 `uint8_t semc_dbi_config_t::tWexHigh_Ns`

39.3.6.0.0.48.11 `uint8_t semc_dbi_config_t::tRdxLow_Ns`

39.3.6.0.0.48.12 `uint8_t semc_dbi_config_t::tRdxHigh_Ns`

39.3.6.0.0.48.13 `uint8_t semc_dbi_config_t::tCsxInterval_Ns`

### 39.3.7 struct `semc_queuea_weight_t`

#### Data Fields

- `uint32_t qos`: 4  
*weight of qos for queue 0.*
- `uint32_t aging`: 4  
*weight of aging for queue 0.*
- `uint32_t slaveHitSwith`: 8  
*weight of read/write switch for queue 0.*
- `uint32_t slaveHitNoswitch`: 8  
*weight of read/write no switch for queue 0.*

**39.3.7.0.0.49 Field Documentation****39.3.7.0.0.49.1** uint32\_t semc\_queuea\_weight\_t::qos**39.3.7.0.0.49.2** uint32\_t semc\_queuea\_weight\_t::aging**39.3.7.0.0.49.3** uint32\_t semc\_queuea\_weight\_t::slaveHitSwith**39.3.7.0.0.49.4** uint32\_t semc\_queuea\_weight\_t::slaveHitNoswitch**39.3.8 struct semc\_queueb\_weight\_t****Data Fields**

- uint32\_t [qos](#): 4  
*weight of qos for queue 1.*
- uint32\_t [aging](#): 4  
*weight of aging for queue 1.*
- uint32\_t [slaveHitSwith](#): 8  
*weight of read/write switch for queue 1.*
- uint32\_t [weightPagehit](#): 8  
*weight of page hit for queue 1 only .*
- uint32\_t [bankRotation](#): 8  
*weight of bank rotation for queue 1 only .*

**39.3.8.0.0.50 Field Documentation****39.3.8.0.0.50.1** uint32\_t semc\_queueb\_weight\_t::qos**39.3.8.0.0.50.2** uint32\_t semc\_queueb\_weight\_t::aging**39.3.8.0.0.50.3** uint32\_t semc\_queueb\_weight\_t::slaveHitSwith**39.3.8.0.0.50.4** uint32\_t semc\_queueb\_weight\_t::weightPagehit**39.3.8.0.0.50.5** uint32\_t semc\_queueb\_weight\_t::bankRotation**39.3.9 struct semc\_axi\_queueweight\_t****Data Fields**

- [semc\\_queuea\\_weight\\_t](#) \* [queueaWeight](#)  
*Weight settings for queue a.*
- [semc\\_queueb\\_weight\\_t](#) \* [queuebWeight](#)  
*Weight settings for queue b.*

## Enumeration Type Documentation

### 39.3.9.0.0.51 Field Documentation

39.3.9.0.0.51.1 `semc_queuea_weight_t* semc_axi_queueweight_t::queueaWeight`

39.3.9.0.0.51.2 `semc_queueb_weight_t* semc_axi_queueweight_t::queuebWeight`

### 39.3.10 struct `semc_config_t`

`busTimeoutCycles`: when `busTimeoutCycles` is zero, the bus timeout cycle is  $255 \times 1024$ . otherwise the bus timeout cycles is `busTimeoutCycles` $\times 1024$ . `cmdTimeoutCycles`: is used for command execution timeout cycles. it's similar to the `busTimeoutCycles`.

## Data Fields

- [semc\\_dqs\\_mode\\_t dqsMode](#)  
*Dummy read strobe mode: use enum in "semc\_dqs\_mode\_t".*
- `uint8_t cmdTimeoutCycles`  
*Command execution timeout cycles.*
- `uint8_t busTimeoutCycles`  
*Bus timeout cycles.*
- [semc\\_axi\\_queueweight\\_t queueWeight](#)  
*AXI queue weight.*

### 39.3.10.0.0.52 Field Documentation

39.3.10.0.0.52.1 `semc_dqs_mode_t semc_config_t::dqsMode`

39.3.10.0.0.52.2 `uint8_t semc_config_t::cmdTimeoutCycles`

39.3.10.0.0.52.3 `uint8_t semc_config_t::busTimeoutCycles`

39.3.10.0.0.52.4 `semc_axi_queueweight_t semc_config_t::queueWeight`

## 39.4 Macro Definition Documentation

39.4.1 `#define FSL_SEMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

## 39.5 Enumeration Type Documentation

39.5.1 `enum _semc_status`

39.5.2 `enum semc_mem_type_t`

Enumerator

*kSEMC\_MemType\_SDRAM* SDRAM.

*kSEMC\_MemType\_SRAM* SRAM.

*kSEMC\_MemType\_NOR* NOR.

*kSEMC\_MemType\_NAND* NAND.  
*kSEMC\_MemType\_8080* 1.

### 39.5.3 enum semc\_waitready\_polarity\_t

Enumerator

*kSEMC\_LowActive* Low active.  
*kSEMC\_HighActive* High active.

### 39.5.4 enum semc\_sdram\_cs\_t

Enumerator

*kSEMC\_SDRAM\_CS0* SEMC SDRAM CS0.  
*kSEMC\_SDRAM\_CS1* SEMC SDRAM CS1.  
*kSEMC\_SDRAM\_CS2* SEMC SDRAM CS2.  
*kSEMC\_SDRAM\_CS3* SEMC SDRAM CS3.

### 39.5.5 enum semc\_nand\_access\_type\_t

### 39.5.6 enum semc\_interrupt\_enable\_t

Enumerator

*kSEMC\_IPCmdDoneInterrupt* Ip command done interrupt.  
*kSEMC\_IPCmdErrInterrupt* Ip command error interrupt.  
*kSEMC\_AXICmdErrInterrupt* AXI command error interrupt.  
*kSEMC\_AXIBusErrInterrupt* AXI bus error interrupt.

### 39.5.7 enum semc\_ipcmd\_datasize\_t

Enumerator

*kSEMC\_IPcmdDataSize\_1bytes* The IP command data size 1 byte.  
*kSEMC\_IPcmdDataSize\_2bytes* The IP command data size 2 byte.  
*kSEMC\_IPcmdDataSize\_3bytes* The IP command data size 3 byte.  
*kSEMC\_IPcmdDataSize\_4bytes* The IP command data size 4 byte.

## Enumeration Type Documentation

### 39.5.8 enum semc\_refresh\_time\_t

Enumerator

*kSEMC\_RefreshThreeClocks* The refresh timing with three bus clocks.

*kSEMC\_RefreshSixClocks* The refresh timing with six bus clocks.

*kSEMC\_RefreshNineClocks* The refresh timing with nine bus clocks.

### 39.5.9 enum semc\_caslatency\_t

Enumerator

*kSEMC\_LatencyOne* Latency 1.

*kSEMC\_LatencyTwo* Latency 2.

*kSEMC\_LatencyThree* Latency 3.

### 39.5.10 enum semc\_sdram\_column\_bit\_num\_t

Enumerator

*kSEMC\_SdramColumn\_12bit* 12 bit.

*kSEMC\_SdramColumn\_11bit* 11 bit.

*kSEMC\_SdramColumn\_10bit* 10 bit.

*kSEMC\_SdramColumn\_9bit* 9 bit.

### 39.5.11 enum sem\_sdram\_burst\_len\_t

Enumerator

*kSEMC\_Sdram\_BurstLen1* Burst length 1.

*kSEMC\_Sdram\_BurstLen2* Burst length 2.

*kSEMC\_Sdram\_BurstLen4* Burst length 4.

*kSEMC\_Sdram\_BurstLen8* Burst length 8.

### 39.5.12 enum semc\_nand\_column\_bit\_num\_t

Enumerator

*kSEMC\_NandColum\_16bit* 16 bit.

*kSEMC\_NandColum\_15bit* 15 bit.

*kSEMC\_NandColum\_14bit* 14 bit.

*kSEMC\_NandColum\_13bit* 13 bit.  
*kSEMC\_NandColum\_12bit* 12 bit.  
*kSEMC\_NandColum\_11bit* 11 bit.  
*kSEMC\_NandColum\_10bit* 10 bit.  
*kSEMC\_NandColum\_9bit* 9 bit.

### 39.5.13 enum sem\_nand\_burst\_len\_t

Enumerator

*kSEMC\_Nand\_BurstLen1* Burst length 1.  
*kSEMC\_Nand\_BurstLen2* Burst length 2.  
*kSEMC\_Nand\_BurstLen4* Burst length 4.  
*kSEMC\_Nand\_BurstLen8* Burst length 8.  
*kSEMC\_Nand\_BurstLen16* Burst length 16.  
*kSEMC\_Nand\_BurstLen32* Burst length 32.  
*kSEMC\_Nand\_BurstLen64* Burst length 64.

### 39.5.14 enum semc\_norsram\_column\_bit\_num\_t

Enumerator

*kSEMC\_NorColum\_12bit* 12 bit.  
*kSEMC\_NorColum\_11bit* 11 bit.  
*kSEMC\_NorColum\_10bit* 10 bit.  
*kSEMC\_NorColum\_9bit* 9 bit.  
*kSEMC\_NorColum\_8bit* 8 bit.  
*kSEMC\_NorColum\_7bit* 7 bit.  
*kSEMC\_NorColum\_6bit* 6 bit.  
*kSEMC\_NorColum\_5bit* 5 bit.  
*kSEMC\_NorColum\_4bit* 4 bit.  
*kSEMC\_NorColum\_3bit* 3 bit.  
*kSEMC\_NorColum\_2bit* 2 bit.

### 39.5.15 enum sem\_norsram\_burst\_len\_t

Enumerator

*kSEMC\_Nor\_BurstLen1* Burst length 1.  
*kSEMC\_Nor\_BurstLen2* Burst length 2.  
*kSEMC\_Nor\_BurstLen4* Burst length 4.

## Enumeration Type Documentation

*kSEMC\_Nor\_BurstLen8* Burst length 8.  
*kSEMC\_Nor\_BurstLen16* Burst length 16.  
*kSEMC\_Nor\_BurstLen32* Burst length 32.  
*kSEMC\_Nor\_BurstLen64* Burst length 64.

### 39.5.16 enum semc\_dbi\_column\_bit\_num\_t

Enumerator

*kSEMC\_Dbi\_Colum\_12bit* 12 bit.  
*kSEMC\_Dbi\_Colum\_11bit* 11 bit.  
*kSEMC\_Dbi\_Colum\_10bit* 10 bit.  
*kSEMC\_Dbi\_Colum\_9bit* 9 bit.  
*kSEMC\_Dbi\_Colum\_8bit* 8 bit.  
*kSEMC\_Dbi\_Colum\_7bit* 7 bit.  
*kSEMC\_Dbi\_Colum\_6bit* 6 bit.  
*kSEMC\_Dbi\_Colum\_5bit* 5 bit.  
*kSEMC\_Dbi\_Colum\_4bit* 4 bit.  
*kSEMC\_Dbi\_Colum\_3bit* 3 bit.  
*kSEMC\_Dbi\_Colum\_2bit* 2 bit.

### 39.5.17 enum sem\_dbi\_burst\_len\_t

Enumerator

*kSEMC\_Dbi\_BurstLen1* Burst length 1.  
*kSEMC\_Dbi\_BurstLen2* Burst length 2.  
*kSEMC\_Dbi\_Dbi\_BurstLen4* Burst length 4.  
*kSEMC\_Dbi\_BurstLen8* Burst length 8.  
*kSEMC\_Dbi\_BurstLen16* Burst length 16.  
*kSEMC\_Dbi\_BurstLen32* Burst length 32.  
*kSEMC\_Dbi\_BurstLen64* Burst length 64.

### 39.5.18 enum semc\_iomux\_pin

Enumerator

*kSEMC\_MUXA8* MUX A8 pin.  
*kSEMC\_MUXCSX0* MUX CSX0 pin.  
*kSEMC\_MUXCSX1* MUX CSX1 Pin.  
*kSEMC\_MUXCSX2* MUX CSX2 Pin.



*kSEMC\_MUXCSX3* MUX CSX3 Pin.  
*kSEMC\_MUXRDY* MUX RDY pin.

### 39.5.19 enum semc\_iomux\_nora27\_pin

Enumerator

*kSEMC\_MORA27\_NONE* No NOR/SRAM A27 pin.  
*kSEMC\_NORA27\_MUXCSX3* MUX CSX3 Pin.  
*kSEMC\_NORA27\_MUXRDY* MUX RDY pin.

### 39.5.20 enum smec\_port\_size\_t

Enumerator

*kSEMC\_PortSize8Bit* 8-Bit port size.  
*kSEMC\_PortSize16Bit* 16-Bit port size.

### 39.5.21 enum semc\_addr\_mode\_t

Enumerator

*kSEMC\_AddrDataMux* SEMC address/data mux mode.  
*kSEMC\_AdvAddrdataMux* Advanced address/data mux mode.  
*kSEMC\_AddrDataNonMux* Address/data non-mux mode.

### 39.5.22 enum semc\_dqs\_mode\_t

Enumerator

*kSEMC\_Loopbackinternal* Dummy read strobe loopbacked internally.  
*kSEMC\_Loopbackdqspad* Dummy read strobe loopbacked from DQS pad.

### 39.5.23 enum semc\_adv\_polarity\_t

Enumerator

*kSEMC\_AdvActiveLow* Adv active low.  
*kSEMC\_AdvActivehigh* Adv active low.

## Enumeration Type Documentation

### 39.5.24 enum semc\_rdy\_polarity\_t

Enumerator

*kSEMC\_RdyActiveLow* Adv active low.  
*kSEMC\_RdyActivehigh* Adv active low.

### 39.5.25 enum semc\_ipcmd\_nand\_addrmode\_t

Enumerator

*kSEMC\_NANDAM\_ColumnRow* Address mode: column and row address(5Byte-CA0/CA1/RA0/-RA1/RA2).  
*kSEMC\_NANDAM\_ColumnCA0* Address mode: column address only(1 Byte-CA0).  
*kSEMC\_NANDAM\_ColumnCA0CA1* Address mode: column address only(2 Byte-CA0/CA1).  
*kSEMC\_NANDAM\_RawRA0* Address mode: row address only(1 Byte-RA0).  
*kSEMC\_NANDAM\_RawRA0RA1* Address mode: row address only(2 Byte-RA0/RA1).  
*kSEMC\_NANDAM\_RawRA0RA1RA2* Address mode: row address only(3 Byte-RA0).

### 39.5.26 enum semc\_ipcmd\_nand\_cmdmode\_t

Enumerator

*kSEMC\_NANDCM\_Command* command.  
*kSEMC\_NANDCM\_CommandHold* Command hold.  
*kSEMC\_NANDCM\_CommandAddress* Command address.  
*kSEMC\_NANDCM\_CommandAddressHold* Command address hold.  
*kSEMC\_NANDCM\_CommandAddressRead* Command address read.  
*kSEMC\_NANDCM\_CommandAddressWrite* Command address write.  
*kSEMC\_NANDCM\_CommandRead* Command read.  
*kSEMC\_NANDCM\_CommandWrite* Command write.  
*kSEMC\_NANDCM\_Read* Read.  
*kSEMC\_NANDCM\_Write* Write.

### 39.5.27 enum semc\_nand\_address\_option\_t

Enumerator

*kSEMC\_NandAddrOption\_5byte\_CA2RA3* CA0+CA1+RA0+RA1+RA2.  
*kSEMC\_NandAddrOption\_4byte\_CA2RA2* CA0+CA1+RA0+RA1.  
*kSEMC\_NandAddrOption\_3byte\_CA2RA1* CA0+CA1+RA0.  
*kSEMC\_NandAddrOption\_4byte\_CA1RA3* CA0+RA0+RA1+RA2.

*kSEMC\_NandAddrOption\_3byte\_CA1RA2* CA0+RA0+RA1.  
*kSEMC\_NandAddrOption\_2byte\_CA1RA1* CA0+RA0.

### 39.5.28 enum semc\_ipcmd\_nor\_dbi\_t

Enumerator

*kSEMC\_NORDBICM\_Read* NOR read.  
*kSEMC\_NORDBICM\_Write* NOR write.

### 39.5.29 enum semc\_ipcmd\_sram\_t

Enumerator

*kSEMC\_SRAMCM\_ArrayRead* SRAM memory array read.  
*kSEMC\_SRAMCM\_ArrayWrite* SRAM memory array write.  
*kSEMC\_SRAMCM\_RegRead* SRAM memory register read.  
*kSEMC\_SRAMCM\_RegWrite* SRAM memory register write.

### 39.5.30 enum semc\_ipcmd\_sdram\_t

Enumerator

*kSEMC\_SDRAMCM\_Read* SDRAM memory read.  
*kSEMC\_SDRAMCM\_Write* SDRAM memory write.  
*kSEMC\_SDRAMCM\_Modeset* SDRAM MODE SET.  
*kSEMC\_SDRAMCM\_Active* SDRAM active.  
*kSEMC\_SDRAMCM\_AutoRefresh* SDRAM auto-refresh.  
*kSEMC\_SDRAMCM\_SelfRefresh* SDRAM self-refresh.  
*kSEMC\_SDRAMCM\_Precharge* SDRAM precharge.  
*kSEMC\_SDRAMCM\_Prechargeall* SDRAM precharge all.

## 39.6 Function Documentation

### 39.6.1 void SEMC\_GetDefaultConfig ( semc\_config\_t \* config )

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

```
semc_config_t config;
SEMC_GetDefaultConfig(&config);
```

## Function Documentation

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | The SEMC configuration structure pointer. |
|---------------|-------------------------------------------|

### 39.6.2 void SEMC\_Init ( SEMC\_Type \* *base*, semc\_config\_t \* *configure* )

This function ungates the SEMC clock and initializes SEMC. This function must be called before calling any other SEMC driver functions.

Parameters

|                  |                                           |
|------------------|-------------------------------------------|
| <i>base</i>      | SEMC peripheral base address.             |
| <i>configure</i> | The SEMC configuration structure pointer. |

### 39.6.3 void SEMC\_Deinit ( SEMC\_Type \* *base* )

This function gates the SEMC clock. As a result, the SEMC module doesn't work after calling this function.

Parameters

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

### 39.6.4 status\_t SEMC\_ConfigureSDRAM ( SEMC\_Type \* *base*, semc\_sdram\_cs\_t *cs*, semc\_sdram\_config\_t \* *config*, uint32\_t *clkSrc\_Hz* )

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>cs</i>        | The chip selection.           |
| <i>config</i>    | The sdram configuration.      |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

### 39.6.5 status\_t SEMC\_ConfigureNAND ( SEMC\_Type \* *base*, semc\_nand\_config\_t \* *config*, uint32\_t *clkSrc\_Hz* )

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>config</i>    | The nand configuration.       |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

**39.6.6** `status_t SEMC_ConfigureNOR ( SEMC_Type * base, semc_nor_config_t * config, uint32_t clkSrc_Hz )`

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>config</i>    | The nor configuration.        |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

**39.6.7** `status_t SEMC_ConfigureSRAM ( SEMC_Type * base, semc_sram_config_t * config, uint32_t clkSrc_Hz )`

Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | SEMC peripheral base address. |
| <i>config</i>    | The sram configuration.       |
| <i>clkSrc_Hz</i> | The SEMC clock frequency.     |

**39.6.8** `status_t SEMC_ConfigureDBI ( SEMC_Type * base, semc_dbi_config_t * config, uint32_t clkSrc_Hz )`

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | SEMC peripheral base address. |
| <i>config</i> | The dbi configuration.        |

## Function Documentation

|                  |                           |
|------------------|---------------------------|
| <i>clkSrc_Hz</i> | The SEMC clock frequency. |
|------------------|---------------------------|

### 39.6.9 static void SEMC\_EnableInterrupts ( SEMC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function enables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [semc\\_interrupt\\_enable\\_t](#). For example, to enable the IP command done and error interrupt, do the following.

```
* SEMC_EnableInterrupts (ENET, kSEMC_IPCmdDoneInterrupt |  
* kSEMC_IPCmdErrInterrupt);
```

#### Parameters

|             |                                                                                                                 |
|-------------|-----------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SEMC peripheral base address.                                                                                   |
| <i>mask</i> | SEMC interrupts to enable. This is a logical OR of the enumeration :: <a href="#">semc_interrupt_enable_t</a> . |

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

This function disables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [semc\\_interrupt\\_enable\\_t](#). For example, to disable the IP command done and error interrupt, do the following.

```
* SEMC_DisableInterrupts (ENET,  
* kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
```

#### Parameters

|             |                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SEMC peripheral base address.                                                                                    |
| <i>mask</i> | SEMC interrupts to disable. This is a logical OR of the enumeration :: <a href="#">semc_interrupt_enable_t</a> . |

### 39.6.11 static bool SEMC\_GetStatusFlag ( SEMC\_Type \* *base* ) [inline], [static]

This function gets the SEMC interrupts event status. User can use the a logical OR of enumeration member as a mask. See [semc\\_interrupt\\_enable\\_t](#).

## Parameters

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

## Returns

status flag, use status flag in `semc_interrupt_enable_t` to get the related status.

**39.6.12 static void SEMC\_ClearStatusFlags ( SEMC\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

The following status register flags can be cleared SEMC interrupt status.

## Parameters

|             |                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------|
| <i>base</i> | SEMC base pointer                                                                                  |
| <i>mask</i> | The status flag mask, a logical OR of enumeration member <a href="#">semc_interrupt_enable_t</a> . |

**39.6.13 static bool SEMC\_IsInIdle ( SEMC\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

True SEMC is in idle, false is not in idle.

**39.6.14 status\_t SEMC\_SendIPCommand ( SEMC\_Type \* *base*, semc\_mem\_type\_t *type*, uint32\_t *address*, uint16\_t *command*, uint32\_t *write*, uint32\_t \* *read* )**

## Parameters

---

## Function Documentation

|                |                                                                                                                                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | SEMC peripheral base address.                                                                                                                                                                                                                                                   |
| <i>type</i>    | SEMC memory type. refer to "semc_mem_type_t"                                                                                                                                                                                                                                    |
| <i>address</i> | SEMC device address.                                                                                                                                                                                                                                                            |
| <i>command</i> | SEMC IP command. For NAND device, we should use the SEMC_BuildNandIP-Command to get the right nand command. For NOR/DBI device, take refer to "semc_ipcmd_nor_dbi_t". For SRAM device, take refer to "semc_ipcmd_sram_t". For SDRAM device, take refer to "semc_ipcmd_sdram_t". |
| <i>write</i>   | Data for write access.                                                                                                                                                                                                                                                          |
| <i>read</i>    | Data pointer for read data out.                                                                                                                                                                                                                                                 |

**39.6.15** `static uint16_t SEMC_BuildNandIPCommand ( uint8_t userCommand, semc_ipcmd_nand_addrmode_t addrMode, semc_ipcmd_nand_cmdmode_t cmdMode ) [inline], [static]`

This function build SEMC NAND IP command. The command is build of user command code, SEMC address mode and SEMC command mode.

Parameters

|                    |                                                           |
|--------------------|-----------------------------------------------------------|
| <i>userCommand</i> | NAND device normal command.                               |
| <i>addrMode</i>    | NAND address mode. Refer to "semc_ipcmd_nand_addrmode_t". |
| <i>cmdMode</i>     | NAND command mode. Refer to "semc_ipcmd_nand_cmdmode_t".  |

**39.6.16** `static bool SEMC_IsNandReady ( SEMC_Type * base ) [inline], [static]`

Parameters

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

Returns

True NAND is ready, false NAND is not ready.

**39.6.17** `status_t SEMC_IPCommandNandWrite ( SEMC_Type * base, uint32_t address, uint8_t * data, uint32_t size_bytes )`



## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | SEMC peripheral base address. |
| <i>address</i>    | SEMC NAND device address.     |
| <i>data</i>       | Data for write access.        |
| <i>size_bytes</i> | Data length.                  |

### 39.6.18 **status\_t SEMC\_IPCommandNandRead ( SEMC\_Type \* *base*, uint32\_t *address*, uint8\_t \* *data*, uint32\_t *size\_bytes* )**

## Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | SEMC peripheral base address.   |
| <i>address</i>    | SEMC NAND device address.       |
| <i>data</i>       | Data pointer for data read out. |
| <i>size_bytes</i> | Data length.                    |

### 39.6.19 **status\_t SEMC\_IPCommandNorWrite ( SEMC\_Type \* *base*, uint32\_t *address*, uint8\_t \* *data*, uint32\_t *size\_bytes* )**

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | SEMC peripheral base address. |
| <i>address</i>    | SEMC NOR device address.      |
| <i>data</i>       | Data for write access.        |
| <i>size_bytes</i> | Data length.                  |

## Function Documentation

**39.6.20** `status_t SEMC_IPCommandNorRead ( SEMC_Type * base, uint32_t address, uint8_t * data, uint32_t size_bytes )`

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | SEMC peripheral base address.   |
| <i>address</i>    | SEMC NOR device address.        |
| <i>data</i>       | Data pointer for data read out. |
| <i>size_bytes</i> | Data length.                    |

---

## Chapter 40

# SNVS: Secure Non-Volatile Storage

### 40.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage (SNVS) module.

The SNVS module is designed to safely hold security-related data such as cryptographic key, time counter, monotonic counter, and general purpose security information. The SNVS includes a low power part, namely SNVS\_LP, that is battery backed by the SVNS (or VBAT) power domain. This enables it to keep this data valid and continue to increment the time counter when the power goes down in the rest of the SoC. The always-powered-up part of the module is isolated from the rest of the logic to ensure that it does not get corrupted when the SoC is powered down. The SNVS is designed to comply with Digital Rights Management (DRM) and other security application rules and requirements. This trusted hardware provides features that allow the system software designer to ensure that the data kept by the device is certifiable. Specially, it incorporates a security monitor that checks for various security conditions. If a security violation is indicated then it invalidates access to its sensitive data, and the secret data, for instance, Zeroizable Secret Key, is zeroized. the SNVS can be also configured to bypass its security features and protection mechanism. In this case it can be used by systems that do not require security.

### Modules

- [Secure Non-Volatile Storage High-Power](#)

### 40.2 Secure Non-Volatile Storage High-Power

#### 40.2.1 Overview

The MCUXpresso SDK provides a Peripheral driver for the Secure Non-Volatile Storage High-Power(S-NVS-HP) module.

The SNVS\_HP is in the chip's power-supply domain and thus receives the power along with the rest of the chip. The SNVS\_HP provides an interface between the SNVS\_LP and the rest of the system; there is no way to access the SNVS\_LP registers except through the SNVS\_HP. For access to the SNVS\_LP registers, the SNVS\_HP must be powered up. It uses a register access permission policy to determine whether the access to the particular registers is permitted.

#### 40.2.2 SNVS\_HP Driver Initialization and Configuration

#### 40.2.3 SNVS\_HP Driver Examples

### Data Structures

- struct `snvs_hp_rtc_datetime_t`  
*Structure is used to hold the date and time. [More...](#)*
- struct `snvs_hp_rtc_config_t`  
*SNVS config structure. [More...](#)*

### Enumerations

- enum `snvs_hp_interrupts_t` {  
    `kSNVS_RTC_AlarmInterrupt` = SNVS\_HPCR\_HPTA\_EN\_MASK,  
    `kSNVS_RTC_PeriodicInterrupt` = SNVS\_HPCR\_PI\_EN\_MASK }  
*List of SNVS interrupts.*
- enum `snvs_hp_status_flags_t` {  
    `kSNVS_RTC_AlarmInterruptFlag` = SNVS\_HPSR\_HPTA\_MASK,  
    `kSNVS_RTC_PeriodicInterruptFlag` = SNVS\_HPSR\_PI\_MASK }  
*List of SNVS flags.*

### Variables

- `uint16_t snvs_hp_rtc_datetime_t::year`  
*Range from 1970 to 2099.*
- `uint8_t snvs_hp_rtc_datetime_t::month`  
*Range from 1 to 12.*
- `uint8_t snvs_hp_rtc_datetime_t::day`

- `uint8_t snvs_hp_rtc_datetime_t::hour`  
Range from 1 to 31 (depending on month).
- `uint8_t snvs_hp_rtc_datetime_t::minute`  
Range from 0 to 23.
- `uint8_t snvs_hp_rtc_datetime_t::second`  
Range from 0 to 59.
- `bool snvs_hp_rtc_config_t::rtcCalEnable`  
*true*: RTC calibration mechanism is enabled; *false*: No calibration is used
- `uint32_t snvs_hp_rtc_config_t::rtcCalValue`  
Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15.
- `uint32_t snvs_hp_rtc_config_t::periodicInterruptFreq`  
Defines frequency of the periodic interrupt; Range from 0 to 15.

## Driver version

- `#define FSL_SNVS_HP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`  
Version 2.0.0.

## Initialization and deinitialization

- `void SNVS_HP_RTC_Init (SNVS_Type *base, const snvs_hp_rtc_config_t *config)`  
*Ungates the SNVS clock and configures the peripheral for basic operation.*
- `void SNVS_HP_RTC_Deinit (SNVS_Type *base)`  
*Stops the RTC and SRTC timers.*
- `void SNVS_HP_RTC_GetDefaultConfig (snvs_hp_rtc_config_t *config)`  
*Fills in the SNVS config struct with the default settings.*

## Non secure RTC current Time & Alarm

- `status_t SNVS_HP_RTC_SetDatetime (SNVS_Type *base, const snvs_hp_rtc_datetime_t *datetime)`  
*Sets the SNVS RTC date and time according to the given time structure.*
- `void SNVS_HP_RTC_GetDatetime (SNVS_Type *base, snvs_hp_rtc_datetime_t *datetime)`  
*Gets the SNVS RTC time and stores it in the given time structure.*
- `status_t SNVS_HP_RTC_SetAlarm (SNVS_Type *base, const snvs_hp_rtc_datetime_t *alarmTime)`  
*Sets the SNVS RTC alarm time.*
- `void SNVS_HP_RTC_GetAlarm (SNVS_Type *base, snvs_hp_rtc_datetime_t *datetime)`  
*Returns the SNVS RTC alarm time.*
- `void SNVS_HP_RTC_TimeSynchronize (SNVS_Type *base)`  
*The function synchronizes RTC counter value with SRTC.*

## Secure Non-Volatile Storage High-Power

### Interrupt Interface

- static void [SNVS\\_HP\\_RTC\\_EnableInterrupts](#) (SNVS\_Type \*base, uint32\_t mask)  
*Enables the selected SNVS interrupts.*
- static void [SNVS\\_HP\\_RTC\\_DisableInterrupts](#) (SNVS\_Type \*base, uint32\_t mask)  
*Disables the selected SNVS interrupts.*
- uint32\_t [SNVS\\_HP\\_RTC\\_GetEnabledInterrupts](#) (SNVS\_Type \*base)  
*Gets the enabled SNVS interrupts.*

### Status Interface

- uint32\_t [SNVS\\_HP\\_RTC\\_GetStatusFlags](#) (SNVS\_Type \*base)  
*Gets the SNVS status flags.*
- static void [SNVS\\_HP\\_RTC\\_ClearStatusFlags](#) (SNVS\_Type \*base, uint32\_t mask)  
*Clears the SNVS status flags.*

### Timer Start and Stop

- static void [SNVS\\_HP\\_RTC\\_StartTimer](#) (SNVS\_Type \*base)  
*Starts the SNVS RTC time counter.*
- static void [SNVS\\_HP\\_RTC\\_StopTimer](#) (SNVS\_Type \*base)  
*Stops the SNVS RTC time counter.*

## 40.2.4 Data Structure Documentation

### 40.2.4.1 struct snvs\_hp\_rtc\_datetime\_t

#### Data Fields

- uint16\_t [year](#)  
*Range from 1970 to 2099.*
- uint8\_t [month](#)  
*Range from 1 to 12.*
- uint8\_t [day](#)  
*Range from 1 to 31 (depending on month).*
- uint8\_t [hour](#)  
*Range from 0 to 23.*
- uint8\_t [minute](#)  
*Range from 0 to 59.*
- uint8\_t [second](#)  
*Range from 0 to 59.*

#### 40.2.4.2 struct snvs\_hp\_rtc\_config\_t

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the SNVS\_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

#### Data Fields

- bool `rtcCalEnable`  
*true: RTC calibration mechanism is enabled; false: No calibration is used*
- uint32\_t `rtcCalValue`  
*Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15.*
- uint32\_t `periodicInterruptFreq`  
*Defines frequency of the periodic interrupt; Range from 0 to 15.*

### 40.2.5 Enumeration Type Documentation

#### 40.2.5.1 enum snvs\_hp\_interrupts\_t

Enumerator

- `kSNVS_RTC_AlarmInterrupt` RTC time alarm.
- `kSNVS_RTC_PeriodicInterrupt` RTC periodic interrupt.

#### 40.2.5.2 enum snvs\_hp\_status\_flags\_t

Enumerator

- `kSNVS_RTC_AlarmInterruptFlag` RTC time alarm flag.
- `kSNVS_RTC_PeriodicInterruptFlag` RTC periodic interrupt flag.

### 40.2.6 Function Documentation

#### 40.2.6.1 void SNVS\_HP\_RTC\_Init ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_config\_t \* *config* )

Note

This API should be called at the beginning of the application using the SNVS driver.

## Secure Non-Volatile Storage High-Power

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | SNVS peripheral base address                        |
| <i>config</i> | Pointer to the user's SNVS configuration structure. |

### 40.2.6.2 void SNVS\_HP\_RTC\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

### 40.2.6.3 void SNVS\_HP\_RTC\_GetDefaultConfig ( snvs\_hp\_rtc\_config\_t \* *config* )

The default values are as follows.

```
* config->rtccalenable = false;  
* config->rtccalvalue = 0U;  
* config->PIFreq = 0U;  
*
```

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>config</i> | Pointer to the user's SNVS configuration structure. |
|---------------|-----------------------------------------------------|

### 40.2.6.4 status\_t SNVS\_HP\_RTC\_SetDatetime ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_datetime\_t \* *datetime* )

Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

Returns

kStatus\_Success: Success in setting the time and starting the SNVS RTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

### 40.2.6.5 void SNVS\_HP\_RTC\_GetDatetime ( SNVS\_Type \* *base*, snvs\_hp\_rtc\_datetime\_t \* *datetime* )



Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

#### 40.2.6.6 **status\_t SNVS\_HP\_RTC\_SetAlarm ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_datetime\_t \* *alarmTime* )**

The function sets the RTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

|                  |                                                          |
|------------------|----------------------------------------------------------|
| <i>base</i>      | SNVS peripheral base address                             |
| <i>alarmTime</i> | Pointer to the structure where the alarm time is stored. |

Returns

kStatus\_Success: success in setting the SNVS RTC alarm  
 kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
 kStatus\_Fail: Error because the alarm time has already passed

#### 40.2.6.7 **void SNVS\_HP\_RTC\_GetAlarm ( SNVS\_Type \* *base*, snvs\_hp\_rtc\_datetime\_t \* *datetime* )**

Parameters

|                 |                                                                            |
|-----------------|----------------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                               |
| <i>datetime</i> | Pointer to the structure where the alarm date and time details are stored. |

#### 40.2.6.8 **void SNVS\_HP\_RTC\_TimeSynchronize ( SNVS\_Type \* *base* )**

Parameters

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

#### 40.2.6.9 **static void SNVS\_HP\_RTC\_EnableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Secure Non-Volatile Storage High-Power

### Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <code>::snvs-_interrupt_enable_t</code> |

**40.2.6.10** `static void SNVS_HP_RTC_DisableInterrupts ( SNVS_Type * base, uint32_t mask ) [inline], [static]`

### Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <code>::snvs-_interrupt_enable_t</code> |

**40.2.6.11** `uint32_t SNVS_HP_RTC_GetEnabledInterrupts ( SNVS_Type * base )`

### Parameters

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

### Returns

The enabled interrupts. This is the logical OR of members of the enumeration `::snvs_interrupt_enable_t`

**40.2.6.12** `uint32_t SNVS_HP_RTC_GetStatusFlags ( SNVS_Type * base )`

### Parameters

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

### Returns

The status flags. This is the logical OR of members of the enumeration `::snvs_status_flags_t`

**40.2.6.13** `static void SNVS_HP_RTC_ClearStatusFlags ( SNVS_Type * base, uint32_t mask ) [inline], [static]`

Parameters

|             |                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                      |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <code>::snvs-_status_flags_t</code> |

**40.2.6.14** `static void SNVS_HP_RTC_StartTimer ( SNVS_Type * base ) [inline], [static]`

Parameters

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

**40.2.6.15** `static void SNVS_HP_RTC_StopTimer ( SNVS_Type * base ) [inline], [static]`

Parameters

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

## 40.2.7 Variable Documentation

**40.2.7.1** `uint16_t snvs_hp_rtc_datetime_t::year`

**40.2.7.2** `uint8_t snvs_hp_rtc_datetime_t::month`

**40.2.7.3** `uint8_t snvs_hp_rtc_datetime_t::day`

**40.2.7.4** `uint8_t snvs_hp_rtc_datetime_t::hour`

**40.2.7.5** `uint8_t snvs_hp_rtc_datetime_t::minute`

**40.2.7.6** `uint8_t snvs_hp_rtc_datetime_t::second`



# Chapter 41

## SPDIF: Sony/Philips Digital Interface

### 41.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Sony/Philips Digital Interface (SPDIF) module of MCUXpresso SDK devices.

SPDIF driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SPDIF initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPDIF peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPDIF functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `spdif_handle_t` as the first parameter. Initialize the handle by calling the [SPDIF\\_TransferTxCreateHandle\(\)](#) or [SPDIF\\_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SPDIF\\_TransferSendNonBlocking\(\)](#) and [SPDIF\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SPDIF_TxIdle` and `kStatus_SPDIF_RxIdle` status.

### 41.2 Typical use case

#### 41.2.1 SPDIF Send/receive using an interrupt method

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

#### 41.2.2 SPDIF Send/receive using a DMA method

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

### Modules

- [SPDIF eDMA Driver](#)

### Data Structures

- struct [spdif\\_config\\_t](#)

## Typical use case

- *SPDIF user configuration structure. [More...](#)*
- struct `spdif_transfer_t`  
*SPDIF transfer structure. [More...](#)*
- struct `spdif_handle_t`  
*SPDIF handle structure. [More...](#)*

## Macros

- #define `SPDIF_XFER_QUEUE_SIZE` (4)  
*SPDIF transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* `spdif_transfer_callback_t`)(SPDIF\_Type \*base, spdif\_handle\_t \*handle, status\_t status, void \*userData)  
*SPDIF transfer callback prototype.*

## Enumerations

- enum `_spdif_status_t` {  
    `kStatus_SPDIF_RxDPLLLocked` = MAKE\_STATUS(kStatusGroup\_SPDIF, 0),  
    `kStatus_SPDIF_TxFIFOError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 1),  
    `kStatus_SPDIF_TxFIFOResync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 2),  
    `kStatus_SPDIF_RxCnew` = MAKE\_STATUS(kStatusGroup\_SPDIF, 3),  
    `kStatus_SPDIF_ValidatyNoGood` = MAKE\_STATUS(kStatusGroup\_SPDIF, 4),  
    `kStatus_SPDIF_RxIllegalSymbol` = MAKE\_STATUS(kStatusGroup\_SPDIF, 5),  
    `kStatus_SPDIF_RxParityBitError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 6),  
    `kStatus_SPDIF_UChannelOverrun` = MAKE\_STATUS(kStatusGroup\_SPDIF, 7),  
    `kStatus_SPDIF_QChannelOverrun` = MAKE\_STATUS(kStatusGroup\_SPDIF, 8),  
    `kStatus_SPDIF_UQChannelSync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 9),  
    `kStatus_SPDIF_UQChannelFrameError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 10),  
    `kStatus_SPDIF_RxFIFOError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 11),  
    `kStatus_SPDIF_RxFIFOResync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 12),  
    `kStatus_SPDIF_LockLoss` = MAKE\_STATUS(kStatusGroup\_SPDIF, 13),  
    `kStatus_SPDIF_TxIdle` = MAKE\_STATUS(kStatusGroup\_SPDIF, 14),  
    `kStatus_SPDIF_RxIdle` = MAKE\_STATUS(kStatusGroup\_SPDIF, 15),  
    `kStatus_SPDIF_QueueFull` = MAKE\_STATUS(kStatusGroup\_SPDIF, 16) }  
*SPDIF return status.*
- enum `spdif_rxfull_select_t` {  
    `kSPDIF_RxFull1Sample` = 0x0u,  
    `kSPDIF_RxFull4Samples`,  
    `kSPDIF_RxFull8Samples`,  
    `kSPDIF_RxFull16Samples` }  
*SPDIF Rx FIFO full flag select, it decides when assert the rx full flag.*
- enum `spdif_txempty_select_t` {  
    `kSPDIF_TxEmpty0Sample` = 0x0u,  
    `kSPDIF_TxEmpty4Samples`,  
    `kSPDIF_TxEmpty8Samples`,

- ```
kSPDIF_TxEmpty12Samples }
    SPDIF tx FIFO EMPTY flag select, it decides when assert the tx empty flag.
```
- enum `spdif_uchannel_source_t` {
 

```
kSPDIF_NoUChannel = 0x0U,
kSPDIF_UChannelFromRx = 0x1U,
kSPDIF_UChannelFromTx = 0x3U }
```

*SPDIF U channel source.*
  - enum `spdif_gain_select_t` {
 

```
kSPDIF_GAIN_24 = 0x0U,
kSPDIF_GAIN_16,
kSPDIF_GAIN_12,
kSPDIF_GAIN_8,
kSPDIF_GAIN_6,
kSPDIF_GAIN_4,
kSPDIF_GAIN_3 }
```

*SPDIF clock gain.*
  - enum `spdif_tx_source_t` {
 

```
kSPDIF_txFromReceiver = 0x1U,
kSPDIF_txNormal = 0x5U }
```

*SPDIF tx data source.*
  - enum `spdif_validity_config_t` {
 

```
kSPDIF_validityFlagAlwaysSet = 0x0U,
kSPDIF_validityFlagAlwaysClear }
```

*SPDIF tx data source.*
  - enum `_spdif_interrupt_enable_t` {
 

```
kSPDIF_RxDPLLLocked = SPDIF_SIE_LOCK_MASK,
kSPDIF_TxFIFOError = SPDIF_SIE_TXUNOV_MASK,
kSPDIF_TxFIFOResync = SPDIF_SIE_TXRESYN_MASK,
kSPDIF_RxControlChannelChange = SPDIF_SIE_CNEW_MASK,
kSPDIF_ValidityFlagNoGood = SPDIF_SIE_VALNOGOOD_MASK,
kSPDIF_RxIllegalSymbol = SPDIF_SIE_SYMERR_MASK,
kSPDIF_RxParityBitError = SPDIF_SIE_BITERR_MASK,
kSPDIF_UChannelReceiveRegisterFull = SPDIF_SIE_URXFUL_MASK,
kSPDIF_UChannelReceiveRegisterOverrun = SPDIF_SIE_URXOV_MASK,
kSPDIF_QChannelReceiveRegisterFull = SPDIF_SIE_QRXFUL_MASK,
kSPDIF_QChannelReceiveRegisterOverrun = SPDIF_SIE_QRXOV_MASK,
kSPDIF_UQChannelSync = SPDIF_SIE_UQSYNC_MASK,
kSPDIF_UQChannelFrameError = SPDIF_SIE_UQERR_MASK,
kSPDIF_RxFIFOError = SPDIF_SIE_RXFIFOUNOV_MASK,
kSPDIF_RxFIFOResync = SPDIF_SIE_RXFIFORESYN_MASK,
kSPDIF_LockLoss = SPDIF_SIE_LOCKLOSS_MASK,
kSPDIF_TxFIFOEmpty = SPDIF_SIE_TXEM_MASK,
kSPDIF_RxFIFOFull = SPDIF_SIE_RXFIFOFUL_MASK }
```

*The SPDIF interrupt enable flag.*
  - enum `_spdif_dma_enable_t` {
 

```
kSPDIF_RxDMAEnable = SPDIF_SCR_DMA_RX_EN_MASK,
```

## Typical use case

```
kSPDIF_TxDMAEnable = SPDIF_SCR_DMA_TX_EN_MASK }  
The DMA request sources.
```

## Driver version

- #define `FSL_SPDIF_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void `SPDIF_Init` (SPDIF\_Type \*base, const `spdif_config_t` \*config)  
*Initializes the SPDIF peripheral.*
- void `SPDIF_GetDefaultConfig` (`spdif_config_t` \*config)  
*Sets the SPDIF configuration structure to default values.*
- void `SPDIF_Deinit` (SPDIF\_Type \*base)  
*De-initializes the SPDIF peripheral.*
- static void `SPDIF_TxFIFOReset` (SPDIF\_Type \*base)  
*Resets the SPDIF Tx.*
- static void `SPDIF_RxFIFOReset` (SPDIF\_Type \*base)  
*Resets the SPDIF Rx.*
- void `SPDIF_TxEnable` (SPDIF\_Type \*base, bool enable)  
*Enables/disables the SPDIF Tx.*
- static void `SPDIF_RxEnable` (SPDIF\_Type \*base, bool enable)  
*Enables/disables the SPDIF Rx.*

## Status

- static uint32\_t `SPDIF_GetStatusFlag` (SPDIF\_Type \*base)  
*Gets the SPDIF status flag state.*
- static void `SPDIF_ClearStatusFlags` (SPDIF\_Type \*base, uint32\_t mask)  
*Clears the SPDIF status flag state.*

## Interrupts

- static void `SPDIF_EnableInterrupts` (SPDIF\_Type \*base, uint32\_t mask)  
*Enables the SPDIF Tx interrupt requests.*
- static void `SPDIF_DisableInterrupts` (SPDIF\_Type \*base, uint32\_t mask)  
*Disables the SPDIF Tx interrupt requests.*

## DMA Control

- static void `SPDIF_EnableDMA` (SPDIF\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the SPDIF DMA requests.*
- static uint32\_t `SPDIF_TxGetLeftDataRegisterAddress` (SPDIF\_Type \*base)  
*Gets the SPDIF Tx left data register address.*
- static uint32\_t `SPDIF_TxGetRightDataRegisterAddress` (SPDIF\_Type \*base)  
*Gets the SPDIF Tx right data register address.*
- static uint32\_t `SPDIF_RxGetLeftDataRegisterAddress` (SPDIF\_Type \*base)  
*Gets the SPDIF Rx left data register address.*
- static uint32\_t `SPDIF_RxGetRightDataRegisterAddress` (SPDIF\_Type \*base)  
*Gets the SPDIF Rx right data register address.*



## Bus Operations

- void [SPDIF\\_TxSetSampleRate](#) (SPDIF\_Type \*base, uint32\_t sampleRate\_Hz, uint32\_t sourceClockFreq\_Hz)  
*Configures the SPDIF Tx sample rate.*
- uint32\_t [SPDIF\\_GetRxSampleRate](#) (SPDIF\_Type \*base, uint32\_t clockSourceFreq\_Hz)  
*Configures the SPDIF Rx audio format.*
- void [SPDIF\\_WriteBlocking](#) (SPDIF\_Type \*base, uint8\_t \*buffer, uint32\_t size)  
*Sends data using a blocking method.*
- static void [SPDIF\\_WriteLeftData](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- static void [SPDIF\\_WriteRightData](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- static void [SPDIF\\_WriteChannelStatusHigh](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- static void [SPDIF\\_WriteChannelStatusLow](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- void [SPDIF\\_ReadBlocking](#) (SPDIF\_Type \*base, uint8\_t \*buffer, uint32\_t size)  
*Receives data using a blocking method.*
- static uint32\_t [SPDIF\\_ReadLeftData](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadRightData](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadChannelStatusHigh](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadChannelStatusLow](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadQChannel](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadUChannel](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*

## Transactional

- void [SPDIF\\_TransferTxCreateHandle](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SPDIF Tx handle.*
- void [SPDIF\\_TransferRxCreateHandle](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SPDIF Rx handle.*
- status\_t [SPDIF\\_TransferSendNonBlocking](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_t](#) \*xfer)  
*Performs an interrupt non-blocking send transfer on SPDIF.*
- status\_t [SPDIF\\_TransferReceiveNonBlocking](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_t](#) \*xfer)  
*Performs an interrupt non-blocking receive transfer on SPDIF.*
- status\_t [SPDIF\\_TransferGetSendCount](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, size\_t \*count)  
*Gets a set byte count.*
- status\_t [SPDIF\\_TransferGetReceiveCount](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, size\_t \*count)  
*Gets a received byte count.*

## Data Structure Documentation

- void [SPDIF\\_TransferAbortSend](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Aborts the current send.*
- void [SPDIF\\_TransferAbortReceive](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Aborts the current IRQ receive.*
- void [SPDIF\\_TransferTxHandleIRQ](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Tx interrupt handler.*
- void [SPDIF\\_TransferRxHandleIRQ](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Tx interrupt handler.*

## 41.3 Data Structure Documentation

### 41.3.1 struct spdif\_config\_t

#### Data Fields

- bool [isTxAutoSync](#)  
*If auto sync mechanism open.*
- bool [isRxAutoSync](#)  
*If auto sync mechanism open.*
- uint8\_t [DPLLClkSource](#)  
*SPDIF DPLL clock source, range from 0~15, meaning is chip-specific.*
- uint8\_t [txClkSource](#)  
*SPDIF tx clock source, range from 0~7, meaning is chip-specific.*
- [spdif\\_rxfull\\_select\\_t rxFullSelect](#)  
*SPDIF rx buffer full select.*
- [spdif\\_txempty\\_select\\_t txFullSelect](#)  
*SPDIF tx buffer empty select.*
- [spdif\\_uchannel\\_source\\_t uChannelSrc](#)  
*U channel source.*
- [spdif\\_tx\\_source\\_t txSource](#)  
*SPDIF tx data source.*
- [spdif\\_validity\\_config\\_t validityConfig](#)  
*Validity flag config.*
- [spdif\\_gain\\_select\\_t gain](#)  
*Rx receive clock measure gain parameter.*

#### 41.3.1.0.53 Field Documentation

##### 41.3.1.0.53.1 [spdif\\_gain\\_select\\_t spdif\\_config\\_t::gain](#)

### 41.3.2 struct spdif\_transfer\_t

#### Data Fields

- uint8\_t \* [data](#)  
*Data start address to transfer.*
- uint8\_t \* [qdata](#)  
*Data buffer for Q channel.*
- uint8\_t \* [udata](#)

- *Data buffer for C channel.*  
size\_t [dataSize](#)  
*Transfer size.*

### 41.3.2.0.0.54 Field Documentation

41.3.2.0.0.54.1 uint8\_t\* [spdif\\_transfer\\_t::data](#)

41.3.2.0.0.54.2 size\_t [spdif\\_transfer\\_t::dataSize](#)

### 41.3.3 struct [\\_spdif\\_handle](#)

#### Data Fields

- uint32\_t [state](#)  
*Transfer status.*
- [spdif\\_transfer\\_callback\\_t](#) [callback](#)  
*Callback function called at transfer event.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*
- [spdif\\_transfer\\_t](#) [spdifQueue](#) [[SPDIF\\_XFER\\_QUEUE\\_SIZE](#)]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [[SPDIF\\_XFER\\_QUEUE\\_SIZE](#)]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*
- uint8\_t [watermark](#)  
*Watermark value.*

## 41.4 Macro Definition Documentation

41.4.1 #define [SPDIF\\_XFER\\_QUEUE\\_SIZE](#) (4)

## 41.5 Enumeration Type Documentation

41.5.1 enum [\\_spdif\\_status\\_t](#)

Enumerator

- [kStatus\\_SPDIF\\_RxDPLLLocked](#) SPDIF Rx PLL locked.*
- [kStatus\\_SPDIF\\_TxFIFOError](#) SPDIF Tx FIFO error.*
- [kStatus\\_SPDIF\\_TxFIFOResync](#) SPDIF Tx left and right FIFO resync.*
- [kStatus\\_SPDIF\\_RxCnew](#) SPDIF Rx status channel value updated.*
- [kStatus\\_SPDIF\\_ValidatyNoGood](#) SPDIF validaty flag not good.*
- [kStatus\\_SPDIF\\_RxIllegalSymbol](#) SPDIF Rx receive illegal symbol.*
- [kStatus\\_SPDIF\\_RxParityBitError](#) SPDIF Rx parity bit error.*
- [kStatus\\_SPDIF\\_UChannelOverrun](#) SPDIF receive U channel overrun.*

## Enumeration Type Documentation

*kStatus\_SPDIF\_QChannelOverrun* SPDIF receive Q channel overrun.  
*kStatus\_SPDIF\_UQChannelSync* SPDIF U/Q channel sync found.  
*kStatus\_SPDIF\_UQChannelFrameError* SPDIF U/Q channel frame error.  
*kStatus\_SPDIF\_RxFIFOError* SPDIF Rx FIFO error.  
*kStatus\_SPDIF\_RxFIFOResync* SPDIF Rx left and right FIFO resync.  
*kStatus\_SPDIF\_LockLoss* SPDIF Rx PLL clock lock loss.  
*kStatus\_SPDIF\_TxIdle* SPDIF Tx is idle.  
*kStatus\_SPDIF\_RxIdle* SPDIF Rx is idle.  
*kStatus\_SPDIF\_QueueFull* SPDIF queue full.

### 41.5.2 enum spdif\_rxfull\_select\_t

Enumerator

*kSPDIF\_RxFull1Sample* Rx full at least 1 sample in left and right FIFO.  
*kSPDIF\_RxFull4Samples* Rx full at least 4 sample in left and right FIFO.  
*kSPDIF\_RxFull8Samples* Rx full at least 8 sample in left and right FIFO.  
*kSPDIF\_RxFull16Samples* Rx full at least 16 sample in left and right FIFO.

### 41.5.3 enum spdif\_txempty\_select\_t

Enumerator

*kSPDIF\_TxEmpty0Sample* Tx empty at most 0 sample in left and right FIFO.  
*kSPDIF\_TxEmpty4Samples* Tx empty at most 4 sample in left and right FIFO.  
*kSPDIF\_TxEmpty8Samples* Tx empty at most 8 sample in left and right FIFO.  
*kSPDIF\_TxEmpty12Samples* Tx empty at most 12 sample in left and right FIFO.

### 41.5.4 enum spdif\_uchannel\_source\_t

Enumerator

*kSPDIF\_NoUChannel* No embedded U channel.  
*kSPDIF\_UChannelFromRx* U channel from receiver, it is CD mode.  
*kSPDIF\_UChannelFromTx* U channel from on chip tx.

### 41.5.5 enum spdif\_gain\_select\_t

Enumerator

*kSPDIF\_GAIN\_24* Gain select is 24.

*kSPDIF\_GAIN\_16* Gain select is 16.  
*kSPDIF\_GAIN\_12* Gain select is 12.  
*kSPDIF\_GAIN\_8* Gain select is 8.  
*kSPDIF\_GAIN\_6* Gain select is 6.  
*kSPDIF\_GAIN\_4* Gain select is 4.  
*kSPDIF\_GAIN\_3* Gain select is 3.

#### 41.5.6 enum spdif\_tx\_source\_t

Enumerator

*kSPDIF\_txFromReceiver* Tx data directly through SPDIF receiver.  
*kSPDIF\_txNormal* Normal operation, data from processor.

#### 41.5.7 enum spdif\_validity\_config\_t

Enumerator

*kSPDIF\_validityFlagAlwaysSet* Outgoing validity flags always set.  
*kSPDIF\_validityFlagAlwaysClear* Outgoing validity flags always clear.

#### 41.5.8 enum \_spdif\_interrupt\_enable\_t

Enumerator

*kSPDIF\_RxDPLLLocked* SPDIF DPLL locked.  
*kSPDIF\_TxFIFOError* Tx FIFO underrun or overrun.  
*kSPDIF\_TxFIFOResync* Tx FIFO left and right channel resync.  
*kSPDIF\_RxControlChannelChange* SPDIF Rx control channel value changed.  
*kSPDIF\_ValidityFlagNoGood* SPDIF validity flag no good.  
*kSPDIF\_RxIllegalSymbol* SPDIF receiver found illegal symbol.  
*kSPDIF\_RxParityBitError* SPDIF receiver found parity bit error.  
*kSPDIF\_UChannelReceiveRegisterFull* SPDIF U channel receive register full.  
*kSPDIF\_UChannelReceiveRegisterOverrun* SPDIF U channel receive register overrun.  
*kSPDIF\_QChannelReceiveRegisterFull* SPDIF Q channel receive register full.  
*kSPDIF\_QChannelReceiveRegisterOverrun* SPDIF Q channel receive register overrun.  
*kSPDIF\_UQChannelSync* SPDIF U/Q channel sync found.  
*kSPDIF\_UQChannelFrameError* SPDIF U/Q channel frame error.  
*kSPDIF\_RxFIFOError* SPDIF Rx FIFO underrun/overrun.  
*kSPDIF\_RxFIFOResync* SPDIF Rx left and right FIFO resync.  
*kSPDIF\_LockLoss* SPDIF receiver loss of lock.

## Function Documentation

*kSPDIF\_TxFIFOEmpty* SPDIF Tx FIFO empty.

*kSPDIF\_RxFIFOFull* SPDIF Rx FIFO full.

### 41.5.9 enum \_spdif\_dma\_enable\_t

Enumerator

*kSPDIF\_RxDMAEnable* Rx FIFO full.

*kSPDIF\_TxDMAEnable* Tx FIFO empty.

## 41.6 Function Documentation

### 41.6.1 void SPDIF\_Init ( SPDIF\_Type \* *base*, const spdif\_config\_t \* *config* )

Ungates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by [SPDIF\\_GetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SPDIF base pointer
<i>config</i>	SPDIF configuration structure.

### 41.6.2 void SPDIF\_GetDefaultConfig ( spdif\_config\_t \* *config* )

This API initializes the configuration structure for use in SPDIF\_Init. The initialized structure can remain unchanged in SPDIF\_Init, or it can be modified before calling SPDIF\_Init. This is an example.

```
spdif_config_t config;  
SPDIF_GetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	-------------------------------------------

#### 41.6.3 void SPDIF\_Deinit ( SPDIF\_Type \* *base* )

This API gates the SPDIF clock. The SPDIF module can't operate unless SPDIF\_Init is called to enable the clock.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

#### 41.6.4 static void SPDIF\_TxFIFOReset ( SPDIF\_Type \* *base* ) [inline], [static]

This function makes Tx FIFO in reset mode.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

#### 41.6.5 static void SPDIF\_RxFIFOReset ( SPDIF\_Type \* *base* ) [inline], [static]

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

#### 41.6.6 void SPDIF\_TxEnable ( SPDIF\_Type \* *base*, bool *enable* )

Parameters

---

## Function Documentation

<i>base</i>	SPDIF base pointer
<i>enable</i>	True means enable SPDIF Tx, false means disable.

**41.6.7 static void SPDIF\_RxEnable ( SPDIF\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer
<i>enable</i>	True means enable SPDIF Rx, false means disable.

**41.6.8 static uint32\_t SPDIF\_GetStatusFlag ( SPDIF\_Type \* *base* )** **[inline],**  
**[static]**

Parameters

<i>base</i>	SPDIF base pointer
-------------	--------------------

Returns

SPDIF status flag value. Use the `_spdif_interrupt_enable_t` to get the status value needed.

**41.6.9 static void SPDIF\_ClearStatusFlags ( SPDIF\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	State mask. It can be a combination of the <code>_spdif_interrupt_enable_t</code> member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits: <ul style="list-style-type: none"><li>• <code>kSPDIF_UChannelReceiveRegisterFull</code></li><li>• <code>kSPDIF_QChannelReceiveRegisterFull</code></li><li>• <code>kSPDIF_TxFIFOEmpty</code></li><li>• <code>kSPDIF_RxFIFOFull</code></li></ul>



**41.6.10** `static void SPDIF_EnableInterrupts ( SPDIF_Type * base, uint32_t mask )`  
`[inline], [static]`

## Function Documentation

### Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSPDIF_WordStartInterruptEnable</li><li>• kSPDIF_SyncErrorInterruptEnable</li><li>• kSPDIF_FIFOWarningInterruptEnable</li><li>• kSPDIF_FIFORequestInterruptEnable</li><li>• kSPDIF_FIFOErrorInterruptEnable</li></ul>

**41.6.11 static void SPDIF\_DisableInterrupts ( SPDIF\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

### Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSPDIF_WordStartInterruptEnable</li><li>• kSPDIF_SyncErrorInterruptEnable</li><li>• kSPDIF_FIFOWarningInterruptEnable</li><li>• kSPDIF_FIFORequestInterruptEnable</li><li>• kSPDIF_FIFOErrorInterruptEnable</li></ul>

**41.6.12 static void SPDIF\_EnableDMA ( SPDIF\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

### Parameters

<i>base</i>	SPDIF base pointer
<i>mask</i>	SPDIF DMA enable mask, The parameter can be a combination of the following sources if defined <ul style="list-style-type: none"><li>• kSPDIF_RxDMAEnable</li><li>• kSPDIF_TxDMAEnable</li></ul>
<i>enable</i>	True means enable DMA, false means disable DMA.

**41.6.13** `static uint32_t SPDIF_TxGetLeftDataRegisterAddress ( SPDIF_Type * base  
 ) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

## Function Documentation

Parameters

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

Returns

data register address.

**41.6.14** `static uint32_t SPDIF_TxGetRightDataRegisterAddress ( SPDIF_Type * base ) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

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

Returns

data register address.

**41.6.15** `static uint32_t SPDIF_RxGetLeftDataRegisterAddress ( SPDIF_Type * base ) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

Parameters

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

Returns

data register address.

**41.6.16** `static uint32_t SPDIF_RxGetRightDataRegisterAddress ( SPDIF_Type * base ) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

## Parameters

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

## Returns

data register address.

#### 41.6.17 void SPDIF\_TxSetSampleRate ( SPDIF\_Type \* *base*, uint32\_t *sampleRate\_Hz*, uint32\_t *sourceClockFreq\_Hz* )

The audio format can be changed at run-time. This function configures the sample rate.

## Parameters

<i>base</i>	SPDIF base pointer.
<i>sampleRate_Hz</i>	SPDIF sample rate frequency in Hz.
<i>sourceClock-Freq_Hz</i>	SPDIF tx clock source frequency in Hz.

#### 41.6.18 uint32\_t SPDIF\_GetRxSampleRate ( SPDIF\_Type \* *base*, uint32\_t *clockSourceFreq\_Hz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

<i>base</i>	SPDIF base pointer.
<i>clockSource-Freq_Hz</i>	SPDIF system clock frequency in hz.

#### 41.6.19 void SPDIF\_WriteBlocking ( SPDIF\_Type \* *base*, uint8\_t \* *buffer*, uint32\_t *size* )

## Note

This function blocks by polling until data is ready to be sent.

## Function Documentation

Parameters

<i>base</i>	SPDIF base pointer.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

**41.6.20 static void SPDIF\_WriteLeftData ( SPDIF\_Type \* *base*, uint32\_t *data* )  
[inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

**41.6.21 static void SPDIF\_WriteRightData ( SPDIF\_Type \* *base*, uint32\_t *data* )  
[inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

**41.6.22 static void SPDIF\_WriteChannelStatusHigh ( SPDIF\_Type \* *base*, uint32\_t  
*data* ) [inline], [static]**

Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

**41.6.23 static void SPDIF\_WriteChannelStatusLow ( SPDIF\_Type \* *base*, uint32\_t  
*data* ) [inline], [static]**

## Parameters

<i>base</i>	SPDIF base pointer.
<i>data</i>	Data needs to be written.

**41.6.24 void SPDIF\_ReadBlocking ( SPDIF\_Type \* *base*, uint8\_t \* *buffer*, uint32\_t *size* )**

## Note

This function blocks by polling until data is ready to be sent.

## Parameters

<i>base</i>	SPDIF base pointer.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

**41.6.25 static uint32\_t SPDIF\_ReadLeftData ( SPDIF\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

Data in SPDIF FIFO.

**41.6.26 static uint32\_t SPDIF\_ReadRightData ( SPDIF\_Type \* *base* ) [inline], [static]**

## Parameters

## Function Documentation

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

Returns

Data in SPDIF FIFO.

**41.6.27** `static uint32_t SPDIF_ReadChannelStatusHigh ( SPDIF_Type * base )`  
`[inline], [static]`

Parameters

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

Returns

Data in SPDIF FIFO.

**41.6.28** `static uint32_t SPDIF_ReadChannelStatusLow ( SPDIF_Type * base )`  
`[inline], [static]`

Parameters

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

Returns

Data in SPDIF FIFO.

**41.6.29** `static uint32_t SPDIF_ReadQChannel ( SPDIF_Type * base )` `[inline],`  
`[static]`

Parameters

---



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

Returns

Data in SPDIF FIFO.

**41.6.30** `static uint32_t SPDIF_ReadUChannel ( SPDIF_Type * base ) [inline], [static]`

Parameters

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

Returns

Data in SPDIF FIFO.

**41.6.31** `void SPDIF_TransferTxCreateHandle ( SPDIF_Type * base, spdif_handle_t * handle, spdif_transfer_callback_t callback, void * userData )`

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function

**41.6.32** `void SPDIF_TransferRxCreateHandle ( SPDIF_Type * base, spdif_handle_t * handle, spdif_transfer_callback_t callback, void * userData )`

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

## Function Documentation

### Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function.

### 41.6.33 **status\_t SPDIF\_TransferSendNonBlocking ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, spdif\_transfer\_t \* *xfer* )**

#### Note

This API returns immediately after the transfer initiates. Call the SPDIF\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SPDIF\_Busy, the transfer is finished.

### Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the spdif_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the <a href="#">spdif_transfer_t</a> structure.

### Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SPDIF_TxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

### 41.6.34 **status\_t SPDIF\_TransferReceiveNonBlocking ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, spdif\_transfer\_t \* *xfer* )**

#### Note

This API returns immediately after the transfer initiates. Call the SPDIF\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SPDIF\_Busy, the transfer is finished.

## Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	Pointer to the <code>spdif_handle_t</code> structure which stores the transfer state.
<i>xfer</i>	Pointer to the <code>spdif_transfer_t</code> structure.

## Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SPDIF_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

#### 41.6.35 `status_t SPDIF_TransferGetSendCount ( SPDIF_Type * base, spdif_handle_t * handle, size_t * count )`

## Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the <code>spdif_handle_t</code> structure which stores the transfer state.
<i>count</i>	Bytes count sent.

## Return values

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

#### 41.6.36 `status_t SPDIF_TransferGetReceiveCount ( SPDIF_Type * base, spdif_handle_t * handle, size_t * count )`

## Parameters

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

## Function Documentation

<i>handle</i>	Pointer to the <code>spdif_handle_t</code> structure which stores the transfer state.
<i>count</i>	Bytes count received.

Return values

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

### 41.6.37 void SPDIF\_TransferAbortSend ( SPDIF\_Type \* *base*, `spdif_handle_t` \* *handle* )

Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the <code>spdif_handle_t</code> structure which stores the transfer state.

### 41.6.38 void SPDIF\_TransferAbortReceive ( SPDIF\_Type \* *base*, `spdif_handle_t` \* *handle* )

Note

This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	Pointer to the <code>spdif_handle_t</code> structure which stores the transfer state.

### 41.6.39 void SPDIF\_TransferTxHandleIRQ ( SPDIF\_Type \* *base*, `spdif_handle_t` \* *handle* )

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the spdif_handle_t structure.

**41.6.40 void SPDIF\_TransferRxHandleIRQ ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle* )**

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	Pointer to the spdif_handle_t structure.

## SPDIF eDMA Driver

### 41.7 SPDIF eDMA Driver

#### 41.7.1 Overview

#### Data Structures

- struct [spdif\\_edma\\_transfer\\_t](#)  
*SPDIF transfer structure. [More...](#)*
- struct [spdif\\_edma\\_handle\\_t](#)  
*SPDIF DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* [spdif\\_edma\\_callback\\_t](#))(SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*SPDIF eDMA transfer callback function for finish and error.*

#### Driver version

- #define [FSL\\_SPDIF\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

#### eDMA Transactional

- void [SPDIF\\_TransferTxCreateHandleEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*dmaLeftHandle, [edma\\_handle\\_t](#) \*dmaRightHandle)  
*Initializes the SPDIF eDMA handle.*
- void [SPDIF\\_TransferRxCreateHandleEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*dmaLeftHandle, [edma\\_handle\\_t](#) \*dmaRightHandle)  
*Initializes the SPDIF Rx eDMA handle.*
- status\_t [SPDIF\\_TransferSendEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPDIF transfer using DMA.*
- status\_t [SPDIF\\_TransferReceiveEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPDIF receive using eDMA.*
- void [SPDIF\\_TransferAbortSendEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle)  
*Aborts a SPDIF transfer using eDMA.*
- void [SPDIF\\_TransferAbortReceiveEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle)  
*Aborts a SPDIF receive using eDMA.*
- status\_t [SPDIF\\_TransferGetSendCountEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count sent by SPDIF.*

- `status_t SPDIF_TransferGetReceiveCountEDMA` (`SPDIF_Type *base, spdif_edma_handle_t *handle, size_t *count`)  
*Gets byte count received by SPDIF.*

## 41.7.2 Data Structure Documentation

### 41.7.2.1 struct `spdif_edma_transfer_t`

#### Data Fields

- `uint8_t * leftData`  
*Data start address to transfer.*
- `uint8_t * rightData`  
*Data start address to transfer.*
- `size_t dataSize`  
*Transfer size.*

#### 41.7.2.1.0.55 Field Documentation

41.7.2.1.0.55.1 `uint8_t* spdif_edma_transfer_t::leftData`

41.7.2.1.0.55.2 `uint8_t* spdif_edma_transfer_t::rightData`

41.7.2.1.0.55.3 `size_t spdif_edma_transfer_t::dataSize`

### 41.7.2.2 struct `_spdif_edma_handle`

#### Data Fields

- `edma_handle_t * dmaLeftHandle`  
*DMA handler for SPDIF left channel.*
- `edma_handle_t * dmaRightHandle`  
*DMA handler for SPDIF right channel.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint8_t count`  
*The transfer data count in a DMA request.*
- `uint32_t state`  
*Internal state for SPDIF eDMA transfer.*
- `spdif_edma_callback_t callback`  
*Callback for users while transfer finish or error occurs.*
- `void * userData`  
*User callback parameter.*
- `edma_tcd_t leftTcd [SPDIF_XFER_QUEUE_SIZE+1U]`  
*TCD pool for eDMA transfer.*
- `edma_tcd_t rightTcd [SPDIF_XFER_QUEUE_SIZE+1U]`  
*TCD pool for eDMA transfer.*
- `spdif_edma_transfer_t spdifQueue [SPDIF_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*

## SPDIF eDMA Driver

- `size_t transferSize` [SPDIF\_XFER\_QUEUE\_SIZE]  
*Data bytes need to transfer, left and right are the same, so use one.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

### 41.7.2.2.0.56 Field Documentation

41.7.2.2.0.56.1 `uint8_t spdif_edma_handle_t::nbytes`

41.7.2.2.0.56.2 `edma_tcd_t spdif_edma_handle_t::leftTcd`[SPDIF\_XFER\_QUEUE\_SIZE+1U]

41.7.2.2.0.56.3 `edma_tcd_t spdif_edma_handle_t::rightTcd`[SPDIF\_XFER\_QUEUE\_SIZE+1U]

41.7.2.2.0.56.4 `spdif_edma_transfer_t spdif_edma_handle_t::spdifQueue`[SPDIF\_XFER\_QUEUE\_SIZE]

41.7.2.2.0.56.5 `volatile uint8_t spdif_edma_handle_t::queueUser`

### 41.7.3 Function Documentation

41.7.3.1 `void SPDIF_TransferTxCreateHandleEDMA ( SPDIF_Type * base,  
spdif_edma_handle_t * handle, spdif_edma_callback_t callback, void *  
userData, edma_handle_t * dmaLeftHandle, edma_handle_t * dmaRightHandle )`

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

Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>base</i>	SPDIF peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>dmaLeftHandle</i>	eDMA handle pointer for left channel, this handle shall be static allocated by users.
<i>dmaRightHandle</i>	eDMA handle pointer for right channel, this handle shall be static allocated by users.



**41.7.3.2 void SPDIF\_TransferRxCreateHandleEDMA ( SPDIF\_Type \* *base*,  
spdif\_edma\_handle\_t \* *handle*, spdif\_edma\_callback\_t *callback*, void \*  
*userData*, edma\_handle\_t \* *dmaLeftHandle*, edma\_handle\_t \* *dmaRightHandle* )**

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

## SPDIF eDMA Driver

### Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>base</i>	SPDIF peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>dmaLeftHandle</i>	eDMA handle pointer for left channel, this handle shall be static allocated by users.
<i>dmaRight-Handle</i>	eDMA handle pointer for right channel, this handle shall be static allocated by users.

### 41.7.3.3 **status\_t SPDIF\_TransferSendEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, spdif\_edma\_transfer\_t \* *xfer* )**

#### Note

This interface returns immediately after the transfer initiates. Call SPDIF\_GetTransferStatus to poll the transfer status and check whether the SPDIF transfer is finished.

### Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

### Return values

<i>kStatus_Success</i>	Start a SPDIF eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_TxBusy</i>	SPDIF is busy sending data.

### 41.7.3.4 **status\_t SPDIF\_TransferReceiveEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, spdif\_edma\_transfer\_t \* *xfer* )**

#### Note

This interface returns immediately after the transfer initiates. Call the SPDIF\_GetReceive-RemainingBytes to poll the transfer status and check whether the SPDIF transfer is finished.

## Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

## Return values

<i>kStatus_Success</i>	Start a SPDIF eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_RxBusy</i>	SPDIF is busy receiving data.

#### 41.7.3.5 void SPDIF\_TransferAbortSendEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle* )

## Parameters

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.

#### 41.7.3.6 void SPDIF\_TransferAbortReceiveEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle* )

## Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF eDMA handle pointer.

#### 41.7.3.7 status\_t SPDIF\_TransferGetSendCountEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

---

## SPDIF eDMA Driver

<i>base</i>	SPDIF base pointer.
<i>handle</i>	SPDIF eDMA handle pointer.
<i>count</i>	Bytes count sent by SPDIF.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

### 41.7.3.8 **status\_t SPDIF\_TransferGetReceiveCountEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

<i>base</i>	SPDIF base pointer
<i>handle</i>	SPDIF eDMA handle pointer.
<i>count</i>	Bytes count received by SPDIF.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

## Chapter 42

# SRC: System Reset Controller Driver

### 42.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Reset Controller (SRC) module.

The System Reset Controller (SRC) controls the reset and boot operation of the SoC. It is responsible for the generation of all reset signals and boot decoding. The reset controller determines the source and the type of reset, such as POR, WARM, COLD, and performs the necessary reset qualification and stretching sequences. Based on the type of reset, the reset logic generates the reset sequence for the entire IC.

### Enumerations

- enum `_src_reset_status_flags` {  
    [kSRC\\_TemperatureSensorResetFlag](#) = SRC\_SRSR\_TSR\_MASK,  
    [kSRC\\_Wdog3ResetFlag](#) = SRC\_SRSR\_WDOG3\_RST\_B\_MASK,  
    [kSRC\\_JTAGSoftwareResetFlag](#) = SRC\_SRSR\_SJC\_MASK,  
    [kSRC\\_JTAGGeneratedResetFlag](#) = SRC\_SRSR\_JTAG\_MASK,  
    [kSRC\\_WatchdogResetFlag](#) = SRC\_SRSR\_WDOG\_MASK,  
    [kSRC\\_IppUserResetFlag](#) = SRC\_SRSR\_IPP\_USER\_RESET\_B\_MASK,  
    [kSRC\\_CsuResetFlag](#) = SRC\_SRSR\_CSU\_RESET\_B\_MASK,  
    [kSRC\\_LockupSysResetFlag](#) = SRC\_SRSR\_LOCKUP\_SYSRESETREQ\_MASK,  
    [kSRC\\_IppResetPinFlag](#) = SRC\_SRSR\_IPP\_RESET\_B\_MASK }  
    *SRC reset status flags.*
- enum `src_warm_reset_bypass_count_t` {  
    [kSRC\\_WarmResetWaitAlways](#) = 0U,  
    [kSRC\\_WarmResetWaitClk16](#) = 1U,  
    [kSRC\\_WarmResetWaitClk32](#) = 2U,  
    [kSRC\\_WarmResetWaitClk64](#) = 3U }  
    *Selection of WARM reset bypass count.*

### Functions

- static void [SRC\\_EnableWDOG3Reset](#) (SRC\_Type \*base, bool enable)  
    *Enable the WDOG3 reset.*
- static void [SRC\\_EnableCoreDebugResetAfterPowerGate](#) (SRC\_Type \*base, bool enable)  
    *Debug reset would be asserted after power gating event.*
- static void [SRC\\_DoSoftwareResetARMCore0](#) (SRC\_Type \*base)  
    *Do software reset the ARM core0 only.*
- static bool [SRC\\_GetSoftwareResetARMCore0Done](#) (SRC\_Type \*base)  
    *Check if the software for ARM core0 is done.*
- static void [SRC\\_EnableWDOGReset](#) (SRC\_Type \*base, bool enable)  
    *Enable the WDOG Reset in SRC.*
- static void [SRC\\_EnableLockupReset](#) (SRC\_Type \*base, bool enable)

## Enumeration Type Documentation

- *Enable the lockup reset.*  
static uint32\_t [SRC\\_GetBootModeWord1](#) (SRC\_Type \*base)  
*Get the boot mode register 1 value.*
- static uint32\_t [SRC\\_GetBootModeWord2](#) (SRC\_Type \*base)  
*Get the boot mode register 2 value.*
- static uint32\_t [SRC\\_GetResetStatusFlags](#) (SRC\_Type \*base)  
*Get the status flags of SRC.*
- void [SRC\\_ClearResetStatusFlags](#) (SRC\_Type \*base, uint32\_t flags)  
*Clear the status flags of SRC.*
- static void [SRC\\_SetGeneralPurposeRegister](#) (SRC\_Type \*base, uint32\_t index, uint32\_t value)  
*Set value to general purpose registers.*
- static uint32\_t [SRC\\_GetGeneralPurposeRegister](#) (SRC\_Type \*base, uint32\_t index)  
*Get the value from general purpose registers.*

## Driver version

- #define [FSL\\_SRC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*SRC driver version 2.0.0.*

## 42.2 Macro Definition Documentation

### 42.2.1 #define FSL\_SRC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 42.3 Enumeration Type Documentation

### 42.3.1 enum \_src\_reset\_status\_flags

Enumerator

- ***kSRC\_TemperatureSensorResetFlag*** Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt need be served before this bit can be cleaned.
- ***kSRC\_Wdog3ResetFlag*** IC Watchdog3 Time-out reset. Indicates whether the reset was the result of the watchdog3 time-out event.
- ***kSRC\_JTAGSoftwareResetFlag*** Indicates whether the reset was the result of setting SJC\_GPCCR bit 31.
- ***kSRC\_JTAGGeneratedResetFlag*** Indicates a reset has been caused by JTAG selection of certain IR codes: EXTEST or HIGHZ.
- ***kSRC\_WatchdogResetFlag*** Indicates a reset has been caused by the watchdog timer timing out. This reset source can be blocked by disabling the watchdog.
- ***kSRC\_IppUserResetFlag*** Indicates whether the reset was the result of the ipp\_user\_reset\_b qualified reset.
- ***kSRC\_CsuResetFlag*** Indicates whether the reset was the result of the csu\_reset\_b input.
- ***kSRC\_LockupSysResetFlag*** Indicates a reset has been caused by CPU lockup or software setting of SYSRESETREQ bit in Application Interrupt and Reset Control Register of the ARM core.
- ***kSRC\_IppResetPinFlag*** Indicates whether reset was the result of ipp\_reset\_b pin (Power-up sequence).

### 42.3.2 enum src\_warm\_reset\_bypass\_count\_t

This type defines the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Enumerator

- kSRC\_WarmResetWaitAlways* System will wait until MMDC acknowledge is asserted.
- kSRC\_WarmResetWaitClk16* Wait 16 32KHz clock cycles before switching the reset.
- kSRC\_WarmResetWaitClk32* Wait 32 32KHz clock cycles before switching the reset.
- kSRC\_WarmResetWaitClk64* Wait 64 32KHz clock cycles before switching the reset.

## 42.4 Function Documentation

### 42.4.1 static void SRC\_EnableWDOG3Reset ( SRC\_Type \* *base*, bool *enable* ) [inline], [static]

The WDOG3 reset is enabled by default.

Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Enable the reset or not.

### 42.4.2 static void SRC\_EnableCoreDebugResetAfterPowerGate ( SRC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Enable the reset or not.

### 42.4.3 static void SRC\_DoSoftwareResetARMCore0 ( SRC\_Type \* *base* ) [inline], [static]

## Function Documentation

### Parameters

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

#### 42.4.4 **static bool SRC\_GetSoftwareResetARMCore0Done ( SRC\_Type \* *base* ) [inline], [static]**

### Parameters

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

### Returns

If the reset is done.

#### 42.4.5 **static void SRC\_EnableWDOGReset ( SRC\_Type \* *base*, bool *enable* ) [inline], [static]**

WDOG Reset is enabled in SRC by default. If the WDOG event to SRC is masked, it would not create a reset to the chip. During the time the WDOG event is masked, when the WDOG event flag is asserted, it would remain asserted regardless of servicing the WDOG module. The only way to clear that bit is the hardware reset.

### Parameters

<i>base</i>	SRC peripheral base address.
<i>enable</i>	Enable the reset or not.

#### 42.4.6 **static void SRC\_EnableLockupReset ( SRC\_Type \* *base*, bool *enable* ) [inline], [static]**

### Parameters

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



<i>enable</i>	Enable the reset or not.
---------------	--------------------------

#### 42.4.7 **static uint32\_t SRC\_GetBootModeWord1 ( SRC\_Type \* *base* ) [inline], [static]**

The Boot Mode register contains bits that reflect the status of BOOT\_CFGx pins of the chip. See to chip-specific document for detail information about value.

Parameters

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

Returns

status of BOOT\_CFGx pins of the chip.

#### 42.4.8 **static uint32\_t SRC\_GetBootModeWord2 ( SRC\_Type \* *base* ) [inline], [static]**

The Boot Mode register contains bits that reflect the status of BOOT\_MODEx Pins and fuse values that controls boot of the chip. See to chip-specific document for detail information about value.

Parameters

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

Returns

status of BOOT\_MODEx Pins and fuse values that controls boot of the chip.

#### 42.4.9 **static uint32\_t SRC\_GetResetStatusFlags ( SRC\_Type \* *base* ) [inline], [static]**

Parameters

---

## Function Documentation

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

Returns

Mask value of status flags, see to [\\_src\\_reset\\_status\\_flags](#).

### 42.4.10 void SRC\_ClearResetStatusFlags ( SRC\_Type \* *base*, uint32\_t *flags* )

Parameters

<i>base</i>	SRC peripheral base address.
<i>Mask</i>	value of status flags to be cleared, see to <a href="#">_src_reset_status_flags</a> .

### 42.4.11 static void SRC\_SetGeneralPurposeRegister ( SRC\_Type \* *base*, uint32\_t *index*, uint32\_t *value* ) [inline], [static]

General purpose registers (GPRx) would hold the value during reset process. Wakeup function could be kept in these register. For example, the GPR1 holds the entry function for waking-up from Partial SLEEP mode while the GPR2 holds the argument. Other GPRx register would store the arbitray values.

Parameters

<i>base</i>	SRC peripheral base address.
<i>index</i>	The index of GPRx register array. Note index 0 reponses the GPR1 register.
<i>value</i>	Setting value for GPRx register.

### 42.4.12 static uint32\_t SRC\_GetGeneralPurposeRegister ( SRC\_Type \* *base*, uint32\_t *index* ) [inline], [static]

Parameters

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

<i>index</i>	The index of GPRx register array. Note index 0 reponses the GPR1 register.
--------------	----------------------------------------------------------------------------

**Returns**

The setting value for GPRx register.



## Chapter 43

# TRNG: True Random Number Generator

### 43.1 Overview

The MCUXpresso SDK provides a peripheral driver for the True Random Number Generator (TRNG) module of MCUXpresso SDK devices.

The True Random Number Generator is a hardware accelerator module that generates a 512-bit entropy as needed by an entropy consuming module or by other post processing functions. A typical entropy consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the TRNG output as its entropy seed. The entropy generated by a TRNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

### 43.2 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.

### 43.3 Get random data from TRNG

1. `TRNG_GetRandomData()` function gets random data from the TRNG module.

This example code shows how to initialize and get random data from the TRNG driver.

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

### Data Structures

- struct `trng_statistical_check_limit_t`  
*Data structure for definition of statistical check limits. [More...](#)*
- struct `trng_config_t`  
*Data structure for the TRNG initialization. [More...](#)*

### Enumerations

- enum `trng_sample_mode_t` {  
    `kTRNG_SampleModeVonNeumann` = 0U,  
    `kTRNG_SampleModeRaw` = 1U,  
    `kTRNG_SampleModeVonNeumannRaw` }  
*TRNG sample mode.*

## Data Structure Documentation

- enum `trng_clock_mode_t` {  
    `kTRNG_ClockModeRingOscillator` = 0U,  
    `kTRNG_ClockModeSystem` = 1U }  
    *TRNG clock mode.*
- enum `trng_ring_osc_div_t` {  
    `kTRNG_RingOscDiv0` = 0U,  
    `kTRNG_RingOscDiv2` = 1U,  
    `kTRNG_RingOscDiv4` = 2U,  
    `kTRNG_RingOscDiv8` = 3U }  
    *TRNG ring oscillator divide.*

## Functions

- status\_t `TRNG_GetDefaultConfig` (`trng_config_t *userConfig`)  
    *Initializes the user configuration structure to default values.*
- status\_t `TRNG_Init` (`TRNG_Type *base`, const `trng_config_t *userConfig`)  
    *Initializes the TRNG.*
- void `TRNG_Deinit` (`TRNG_Type *base`)  
    *Shuts down the TRNG.*
- status\_t `TRNG_GetRandomData` (`TRNG_Type *base`, void \*data, size\_t dataSize)  
    *Gets random data.*

## Driver version

- #define `FSL_TRNG_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
    *TRNG driver version 2.0.2.*

## 43.4 Data Structure Documentation

### 43.4.1 struct `trng_statistical_check_limit_t`

Used by `trng_config_t`.

## Data Fields

- uint32\_t `maximum`  
    *Maximum limit.*
- uint32\_t `minimum`  
    *Minimum limit.*

#### 43.4.1.0.0.57 Field Documentation

43.4.1.0.0.57.1 `uint32_t trng_statistical_check_limit_t::maximum`

43.4.1.0.0.57.2 `uint32_t trng_statistical_check_limit_t::minimum`

#### 43.4.2 struct `trng_config_t`

This structure initializes the TRNG by calling the `TRNG_Init()` function. It contains all TRNG configurations.

#### Data Fields

- `bool lock`  
*Disable programmability of TRNG registers.*
- `trng_clock_mode_t clockMode`  
*Clock mode used to operate TRNG.*
- `trng_ring_osc_div_t ringOscDiv`  
*Ring oscillator divide used by TRNG.*
- `trng_sample_mode_t sampleMode`  
*Sample mode of the TRNG ring oscillator.*
- `uint16_t entropyDelay`  
*Entropy Delay.*
- `uint16_t sampleSize`  
*Sample Size.*
- `uint16_t sparseBitLimit`  
*Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated.*
- `uint8_t retryCount`  
*Retry count.*
- `uint8_t longRunMaxLimit`  
*Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.*
- `trng_statistical_check_limit_t monobitLimit`  
*Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.*
- `trng_statistical_check_limit_t runBit1Limit`  
*Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.*
- `trng_statistical_check_limit_t runBit2Limit`  
*Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.*
- `trng_statistical_check_limit_t runBit3Limit`  
*Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.*
- `trng_statistical_check_limit_t runBit4Limit`  
*Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.*
- `trng_statistical_check_limit_t runBit5Limit`

## Data Structure Documentation

*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.*

### 43.4.2.0.0.58 Field Documentation

**43.4.2.0.0.58.1** `bool trng_config_t::lock`

**43.4.2.0.0.58.2** `trng_clock_mode_t trng_config_t::clockMode`

**43.4.2.0.0.58.3** `trng_ring_osc_div_t trng_config_t::ringOscDiv`

**43.4.2.0.0.58.4** `trng_sample_mode_t trng_config_t::sampleMode`

**43.4.2.0.0.58.5** `uint16_t trng_config_t::entropyDelay`

Defines the length (in system clocks) of each Entropy sample taken.

**43.4.2.0.0.58.6** `uint16_t trng_config_t::sampleSize`

Defines the total number of Entropy samples that will be taken during Entropy generation.

**43.4.2.0.0.58.7** `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.

**43.4.2.0.0.58.8** `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.



- 43.4.2.0.0.58.9 `uint8_t trng_config_t::longRunMaxLimit`
- 43.4.2.0.0.58.10 `trng_statistical_check_limit_t trng_config_t::monobitLimit`
- 43.4.2.0.0.58.11 `trng_statistical_check_limit_t trng_config_t::runBit1Limit`
- 43.4.2.0.0.58.12 `trng_statistical_check_limit_t trng_config_t::runBit2Limit`
- 43.4.2.0.0.58.13 `trng_statistical_check_limit_t trng_config_t::runBit3Limit`
- 43.4.2.0.0.58.14 `trng_statistical_check_limit_t trng_config_t::runBit4Limit`
- 43.4.2.0.0.58.15 `trng_statistical_check_limit_t trng_config_t::runBit5Limit`
- 43.4.2.0.0.58.16 `trng_statistical_check_limit_t trng_config_t::runBit6PlusLimit`
- 43.4.2.0.0.58.17 `trng_statistical_check_limit_t trng_config_t::pokerLimit`
- 43.4.2.0.0.58.18 `trng_statistical_check_limit_t trng_config_t::frequencyCountLimit`

## 43.5 Macro Definition Documentation

### 43.5.1 `#define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

Current version: 2.0.2

Change log:

- Version 2.0.2
  - fix MISRA issues
- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

## 43.6 Enumeration Type Documentation

### 43.6.1 `enum trng_sample_mode_t`

Used by [trng\\_config\\_t](#).

Enumerator

- kTRNG\_SampleModeVonNeumann* Use von Neumann data in both Entropy shifter and Statistical Checker.
- kTRNG\_SampleModeRaw* Use raw data into both Entropy shifter and Statistical Checker.
- kTRNG\_SampleModeVonNeumannRaw* Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

## Function Documentation

### 43.6.2 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.

### 43.6.3 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.

## 43.7 Function Documentation

### 43.7.1 status\_t TRNG\_GetDefaultConfig ( trng\_config\_t \* userConfig )

This function initializes the configuration structure to default values. The default values are as follows.

```
* user_config->lock = 0;
* user_config->clockMode = kTRNG_ClockModeRingOscillator;
* user_config->ringOscDiv = kTRNG_RingOscDiv0; Or to other kTRNG_RingOscDiv[2|8]
  depending on the 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;  
*
```

## Function Documentation

### Parameters

<i>user_config</i>	User configuration structure.
--------------------	-------------------------------

### Returns

If successful, returns the `kStatus_TRNG_Success`. Otherwise, it returns an error.

### 43.7.2 `status_t TRNG_Init ( TRNG_Type * base, const trng_config_t * userConfig )`

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

### Parameters

<i>base</i>	TRNG base address
<i>userConfig</i>	Pointer to the initialization configuration structure.

### Returns

If successful, returns the `kStatus_TRNG_Success`. Otherwise, it returns an error.

### 43.7.3 `void TRNG_Deinit ( TRNG_Type * base )`

This function shuts down the TRNG.

### Parameters

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

### 43.7.4 `status_t TRNG_GetRandomData ( TRNG_Type * base, void * data, size_t dataSize )`

This function gets random data from the TRNG.

### Parameters

---

<i>base</i>	TRNG base address.
<i>data</i>	Pointer address used to store random data.
<i>dataSize</i>	Size of the buffer pointed by the data parameter.

Returns

random data



## Chapter 44

# TSC: Touch Screen Controller Driver

### 44.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Touch Screen Controller(TSC) module of MCUXpresso SDK devices.

### 44.2 Typical use case

#### 44.2.1 4-wire Polling Configuration

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

#### 44.2.2 4-wire Interrupt Configuration

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

### Data Structures

- struct `tsc_config_t`  
@ Controller configuration. [More...](#)

### Macros

- #define `FSL_TSC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*TSC driver version.*

### Enumerations

- enum `tsc_detection_mode_t` {  
    `kTSC_Detection4WireMode` = 0U,  
    `kTSC_Detection5WireMode` = 1U }  
    @ Controller detection mode.
- enum `tsc_corrdinate_value_selection_t` {  
    `kTSC_XCoordinateValueSelection` = 0U,  
    `kTSC_YCoordinateValueSelection` = 1U }  
    @ Coordinate value mask.

## Typical use case

- enum `_tsc_interrupt_signal_mask` {  
    `kTSC_IdleSoftwareSignalEnable` = `TSC_INT_SIG_EN_IDLE_SW_SIG_EN_MASK`,  
    `kTSC_ValidSignalEnable`,  
    `kTSC_DetectSignalEnable`,  
    `kTSC_MeasureSignalEnable` = `TSC_INT_SIG_EN_MEASURE_SIG_EN_MASK` }  
    @ *Interrupt signal enable/disable mask.*
- enum `_tsc_interrupt_mask` {  
    `kTSC_IdleSoftwareInterruptEnable`,  
    `kTSC_DetectInterruptEnable`,  
    `kTSC_MeasureInterruptEnable` = `TSC_INT_EN_MEASURE_INT_EN_MASK` }  
    @ *Interrupt enable/disable mask.*
- enum `_tsc_interrupt_status_flag_mask` {  
    `kTSC_IdleSoftwareFlag`,  
    `kTSC_ValidSignalFlag`,  
    `kTSC_DetectSignalFlag` = `TSC_INT_STATUS_DETECT_MASK`,  
    `kTSC_MeasureSignalFlag` }  
    @ *Interrupt Status flag mask.*
- enum `_tsc_adc_status_flag_mask` {  
    `kTSC_ADCCOCOSignalFlag`,  
    `kTSC_ADCCConversionValueFlag` }  
    @ *ADC status flag mask.*
- enum `_tsc_status_flag_mask` {  
    `kTSC_IntermediateStateFlag` = `TSC_DEBUG_MODE2_INTERMEDIATE_MASK`,  
    `kTSC_DetectFiveWireFlag` = `TSC_DEBUG_MODE2_DETECT_FIVE_WIRE_MASK`,  
    `kTSC_DetectFourWireFlag` = `TSC_DEBUG_MODE2_DETECT_FOUR_WIRE_MASK`,  
    `kTSC_GlitchThresholdFlag` = `TSC_DEBUG_MODE2_DE_GLITCH_MASK`,  
    `kTSC_StateMachineFlag` }  
    @ *TSC status flag mask.*
- enum `tsc_state_machine_t` {  
    `kTSC_IdleState` = `0U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT`,  
    `kTSC_1stPreChargeState` = `1U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT`,  
    `kTSC_1stDetectState` = `2U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT`,  
    `kTSC_XMeasureState` = `3U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT`,  
    `kTSC_YMeasureState` = `4U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT`,  
    `kTSC_2ndPreChargeState` = `5U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT`,  
    `kTSC_2ndDetectState` = `6U << TSC_DEBUG_MODE2_STATE_MACHINE_SHIFT` }  
    @ *TSC state machine.*
- enum `tsc_glitch_threshold_t` {  
    `kTSC_glitchThresholdALT0`,  
    `kTSC_glitchThresholdALT1`,  
    `kTSC_glitchThresholdALT2`,  
    `kTSC_glitchThresholdALT3` }  
    @ *TSC glitch threshold.*
- enum `tsc_trigger_signal_t` {



```

kTSC_TriggerToChannel0 = 1U << 0U,
kTSC_TriggerToChannel1 = 1U << 1U,
kTSC_TriggerToChannel2 = 1U << 2U,
kTSC_TriggerToChannel3 = 1U << 3U,
kTSC_TriggerToChannel4 = 1U << 4U }

```

@ Hardware trigger select signal, select which ADC channel to start conversion.

- enum `tsc_port_source_t` {
 

```

kTSC_WiperPortSource = 0U,
kTSC_YnlrPortSource = 1U,
kTSC_YpllPortSource = 2U,
kTSC_XnurPortSource = 3U,
kTSC_XpulPortSource = 4U }

```

@ TSC controller ports.
- enum `tsc_port_mode_t` {
 

```

kTSC_PortOffMode = 0U,
kTSC_Port200k_PullUpMode = 1U << 2U,
kTSC_PortPullUpMode = 1U << 1U,
kTSC_PortPullDownMode = 1U << 0U }

```

@ TSC port mode.

## Functions

- void `TSC_Init` (TSC\_Type \*base, const `tsc_config_t` \*config)
 

*Initialize the TSC module.*
- void `TSC_Deinit` (TSC\_Type \*base)
 

*De-initializes the TSC module.*

## Variables

- bool `tsc_config_t::enableAutoMeasure`

*Enable the auto-measure.*
- uint32\_t `tsc_config_t::measureDelayTime`

*Set delay time(0U~0xFFFFFFFFU) to even potential distribution ready.It is a preparation for measure stage.*
- uint32\_t `tsc_config_t::prechargeTime`

*Set pre-charge time(1U~0xFFFFFFFFU) to make the upper layer of screen to charge to positive high.*
- `tsc_detection_mode_t` `tsc_config_t::detectionMode`

*Select the detection mode.*

## 44.3 Data Structure Documentation

### 44.3.1 struct `tsc_config_t`

#### Data Fields

- bool `enableAutoMeasure`

*Enable the auto-measure.*
- uint32\_t `measureDelayTime`

*Set delay time(0U~0xFFFFFFFFU) to even potential distribution ready.It is a preparation for measure stage.*

## Enumeration Type Documentation

- uint32\_t [prechargeTime](#)  
Set pre-charge time(1U~0xFFFFFFFFU) to make the upper layer of screen to charge to positive high.
- [tsc\\_detection\\_mode\\_t](#) `detectionMode`  
Select the detection mode.

### 44.4 Macro Definition Documentation

#### 44.4.1 #define FSL\_TSC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

### 44.5 Enumeration Type Documentation

#### 44.5.1 enum tsc\_detection\_mode\_t

Enumerator

*kTSC\_Detection4WireMode* 4-Wire Detection Mode.  
*kTSC\_Detection5WireMode* 5-Wire Detection Mode.

#### 44.5.2 enum tsc\_corrordinate\_value\_selection\_t

Enumerator

*kTSC\_XCoordinateValueSelection* X coordinate value is selected.  
*kTSC\_YCoordinateValueSelection* Y coordinate value is selected.

#### 44.5.3 enum \_tsc\_interrupt\_signal\_mask

Enumerator

*kTSC\_IdleSoftwareSignalEnable* Enable the interrupt signal when the controller has return to idle status. The signal is only valid after using TSC\_ReturnToIdleStatus API.  
*kTSC\_ValidSignalEnable* Enable the interrupt signal when controller receives a detect signal after measurement.  
*kTSC\_DetectSignalEnable* Enable the interrupt signal when controller receives a detect signal.  
*kTSC\_MeasureSignalEnable* Enable the interrupt signal after the touch detection which follows measurement.

#### 44.5.4 enum\_tsc\_interrupt\_mask

Enumerator

***kTSC\_IdleSoftwareInterruptEnable*** Enable the interrupt when the controller has return to idle status. The interrupt is only valid after using TSC\_ReturnToIdleStatus API.

***kTSC\_DetectInterruptEnable*** Enable the interrupt when controller receive a detect signal.

***kTSC\_MeasureInterruptEnable*** Enable the interrupt after the touch detection which follows measurement.

#### 44.5.5 enum\_tsc\_interrupt\_status\_flag\_mask

Enumerator

***kTSC\_IdleSoftwareFlag*** This flag is set if the controller has return to idle status. The flag is only valid after using TSC\_ReturnToIdleStatus API.

***kTSC\_ValidSignalFlag*** This flag is set if controller receives a detect signal after measurement.

***kTSC\_DetectSignalFlag*** This flag is set if controller receives a detect signal.

***kTSC\_MeasureSignalFlag*** This flag is set after the touch detection which follows measurement.  
Note: Valid signal flag will be cleared along with measure signal flag.

#### 44.5.6 enum\_tsc\_adc\_status\_flag\_mask

Enumerator

***kTSC\_ADCCOCOSignalFlag*** This signal is generated by ADC when a conversion is completed.

***kTSC\_ADCCONVERSIONVALUEFLAG*** This signal is generated by ADC and indicates the result of an ADC conversion.

#### 44.5.7 enum\_tsc\_status\_flag\_mask

Enumerator

***kTSC\_IntermediateStateFlag*** This flag is set if TSC is in intermediate state, between two state machine states.

***kTSC\_DetectFiveWireFlag*** This flag is set if TSC receives a 5-wire detect signal. It is only valid when the TSC in detect state and DETECT\_ENABLE\_FIVE\_WIRE bit is set.

***kTSC\_DetectFourWireFlag*** This flag is set if TSC receives a 4-wire detect signal. It is only valid when the TSC in detect state and DETECT\_ENABLE\_FOUR\_WIRE bit is set.

***kTSC\_GlitchThresholdFlag*** This field indicates glitch threshold. The threshold is defined by number of clock cycles. See "tsc\_glitch\_threshold\_t". If value = 00, Normal function: 0x1fff ipg clock cycles, Low power mode: 0x9 low power clock cycles. If value = 01, Normal function: 0xffff ipg

## Enumeration Type Documentation

clock cycles, Low power mode: :0x7 low power clock cycles. If value = 10, Normal function: 0x7ff ipg clock cycles, Low power mode:0x5 low power clock cycles. If value = 11, Normal function: 0x3 ipg clock cycles, Low power mode:0x3 low power clock cycles.

***kTSC\_StateMachineFlag*** This field indicates the state of TSC. See "tsc\_state\_machine\_t"; if value = 000, Controller is in idle state. if value = 001, Controller is in 1st-Pre-charge state. if value = 010, Controller is in 1st-detect state. if value = 011, Controller is in x-measure state. if value = 100, Controller is in y-measure state. if value = 101, Controller is in 2nd-Pre-charge state. if value = 110, Controller is in 2nd-detect state.

### 44.5.8 enum tsc\_state\_machine\_t

These seven states are TSC complete workflow.

Enumerator

***kTSC\_IdleState*** Controller is in idle state.

***kTSC\_1stPreChargeState*** Controller is in 1st-Pre-charge state.

***kTSC\_1stDetectState*** Controller is in 1st-detect state.

***kTSC\_XMeasureState*** Controller is in x-measure state.

***kTSC\_YMeasureState*** Controller is in y-measure state.

***kTSC\_2ndPreChargeState*** Controller is in 2nd-Pre-charge state.

***kTSC\_2ndDetectState*** Controller is in 2nd-detect state.

### 44.5.9 enum tsc\_glitch\_threshold\_t

Enumerator

***kTSC\_glitchThresholdALT0*** Normal function: 0x1fff ipg clock cycles, Low power mode: 0x9 low power clock cycles.

***kTSC\_glitchThresholdALT1*** Normal function: 0xffff ipg clock cycles, Low power mode: :0x7 low power clock cycles.

***kTSC\_glitchThresholdALT2*** Normal function: 0x7ff ipg clock cycles, Low power mode: :0x5 low power clock cycles.

***kTSC\_glitchThresholdALT3*** Normal function: 0x3 ipg clock cycles, Low power mode: :0x3 low power clock cycles.

### 44.5.10 enum tsc\_trigger\_signal\_t

Enumerator

***kTSC\_TriggerToChannel0*** Trigger to ADC channel0. ADC\_HC0 register will be used to conversion.

***kTSC\_TriggerToChannel1*** Trigger to ADC channel1. ADC\_HC1 register will be used to conversion.

***kTSC\_TriggerToChannel2*** Trigger to ADC channel2. ADC\_HC2 register will be used to conversion.

***kTSC\_TriggerToChannel3*** Trigger to ADC channel3. ADC\_HC3 register will be used to conversion.

***kTSC\_TriggerToChannel4*** Trigger to ADC channel4. ADC\_HC4 register will be used to conversion.

#### 44.5.11 enum tsc\_port\_source\_t

Enumerator

***kTSC\_WiperPortSource*** TSC controller wiper port.

***kTSC\_YnlrPortSource*** TSC controller ynlr port.

***kTSC\_YpllPortSource*** TSC controller ypll port.

***kTSC\_XnurPortSource*** TSC controller xnur port.

***kTSC\_XpulPortSource*** TSC controller xpul port.

#### 44.5.12 enum tsc\_port\_mode\_t

Enumerator

***kTSC\_PortOffMode*** Disable pull up/down mode.

***kTSC\_Port200k\_PullUpMode*** 200k-pull up mode.

***kTSC\_PortPullUpMode*** Pull up mode.

***kTSC\_PortPullDownMode*** Pull down mode.

### 44.6 Function Documentation

#### 44.6.1 void TSC\_Init ( TSC\_Type \* *base*, const tsc\_config\_t \* *config* )

Parameters

<i>base</i>	TSC peripheral base address.
<i>config</i>	Pointer to "tsc_config_t" structure.

#### 44.6.2 void TSC\_Deinit ( TSC\_Type \* *base* )

## Variable Documentation

Parameters

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

### 44.7 Variable Documentation

#### 44.7.1 `bool tsc_config_t::enableAutoMeasure`

It indicates after detect touch, whether automatic start measurement

#### 44.7.2 `uint32_t tsc_config_t::measureDelayTime`

If measure delay time is too short, maybe it would have an undesired effect on measure value.

#### 44.7.3 `uint32_t tsc_config_t::prechargeTime`

It is a preparation for detection stage. Pre-charge time must is greater than 0U, otherwise TSC could not work normally. If pre-charge delay time is too short, maybe it would have an undesired effect on generation of valid signal(`kTSC_ValidSignalFlag`).

#### 44.7.4 `tsc_detection_mode_t tsc_config_t::detectionMode`

See "`tsc_detection_mode_t`".

# Chapter 45

## USDHC: ultra Secured Digital Host Controller Driver

### 45.1 Overview

The MCUXpresso SDK provides a peripheral driver for the ultra Secured Digital Host Controller (USDHC) module of MCUXpresso SDK/i.MX devices.

### 45.2 Typical use case

#### 45.2.1 USDHC Operation

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

### Data Structures

- struct `usdhc_adma2_descriptor_t`  
*Defines the ADMA2 descriptor structure. [More...](#)*
- struct `usdhc_capability_t`  
*USDHC capability information. [More...](#)*
- struct `usdhc_boot_config_t`  
*Data structure to configure the MMC boot feature. [More...](#)*
- struct `usdhc_config_t`  
*Data structure to initialize the USDHC. [More...](#)*
- struct `usdhc_data_t`  
*Card data descriptor. [More...](#)*
- struct `usdhc_command_t`  
*Card command descriptor. [More...](#)*
- struct `usdhc_adma_config_t`  
*ADMA configuration. [More...](#)*
- struct `usdhc_transfer_t`  
*Transfer state. [More...](#)*
- struct `usdhc_transfer_callback_t`  
*USDHC callback functions. [More...](#)*
- struct `usdhc_handle_t`  
*USDHC handle. [More...](#)*
- struct `usdhc_host_t`  
*USDHC host descriptor. [More...](#)*

### Macros

- #define `USDHC_MAX_BLOCK_COUNT` (`USDHC_BLK_ATT_BLKCNT_MASK >> USDHC_BLK_ATT_BLKCNT_SHIFT`)  
*Maximum block count can be set one time.*
- #define `USDHC_ADMA1_ADDRESS_ALIGN` (4096U)  
*The alignment size for ADDRESS filed in ADMA1's descriptor.*

## Typical use case

- #define `USDHC_ADMA1_LENGTH_ALIGN` (4096U)  
*The alignment size for LENGTH field in ADMA1's descriptor.*
- #define `USDHC_ADMA2_ADDRESS_ALIGN` (4U)  
*The alignment size for ADDRESS field in ADMA2's descriptor.*
- #define `USDHC_ADMA2_LENGTH_ALIGN` (4U)  
*The alignment size for LENGTH field in ADMA2's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT` (12U)  
*The bit shift for ADDRESS field in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK` (0xFFFFFU)  
*The bit mask for ADDRESS field in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT` (12U)  
*The bit shift for LENGTH field in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK` (0xFFFFU)  
*The mask for LENGTH field in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (`USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK` - 3U)  
*The maximum value of LENGTH field in ADMA1's descriptor.*
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT` (16U)  
*The bit shift for LENGTH field in ADMA2's descriptor.*
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK` (0xFFFFU)  
*The bit mask for LENGTH field in ADMA2's descriptor.*
- #define `USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (`USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK` - 3U)  
*The maximum value of LENGTH field in ADMA2's descriptor.*

## Typedefs

- typedef `uint32_t usdhc_adma1_descriptor_t`  
*Defines the adma1 descriptor structure.*
- typedef `status_t(* usdhc_transfer_function_t)(USDHC_Type *base, usdhc_transfer_t *content)`  
*USDHC transfer function.*

## Enumerations

- enum `_usdhc_status` {  
`kStatus_USDHC_BusyTransferring` = MAKE\_STATUS(kStatusGroup\_USDHC, 0U),  
`kStatus_USDHC_PrepareAdmaDescriptorFailed` = MAKE\_STATUS(kStatusGroup\_USDHC, 1U),  
`kStatus_USDHC_SendCommandFailed` = MAKE\_STATUS(kStatusGroup\_USDHC, 2U),  
`kStatus_USDHC_TransferDataFailed` = MAKE\_STATUS(kStatusGroup\_USDHC, 3U),  
`kStatus_USDHC_DMADDataAddrNotAlign` = MAKE\_STATUS(kStatusGroup\_USDHC, 4U),  
`kStatus_USDHC_ReTuningRequest` = MAKE\_STATUS(kStatusGroup\_USDHC, 5U),  
`kStatus_USDHC_TuningError` = MAKE\_STATUS(kStatusGroup\_USDHC, 6U),  
`kStatus_USDHC_NotSupport` = MAKE\_STATUS(kStatusGroup\_USDHC, 7U) }  
*USDHC status.*
- enum `_usdhc_capability_flag` {



```

kUSDHC_SupportAdmaFlag = USDHC_HOST_CTRL_CAP_ADMAS_MASK,
kUSDHC_SupportHighSpeedFlag = USDHC_HOST_CTRL_CAP_HSS_MASK,
kUSDHC_SupportDmaFlag = USDHC_HOST_CTRL_CAP_DMAS_MASK,
kUSDHC_SupportSuspendResumeFlag = USDHC_HOST_CTRL_CAP_SRS_MASK,
kUSDHC_SupportV330Flag = USDHC_HOST_CTRL_CAP_VS33_MASK,
kUSDHC_SupportV300Flag = USDHC_HOST_CTRL_CAP_VS30_MASK,
kUSDHC_SupportV180Flag = USDHC_HOST_CTRL_CAP_VS18_MASK,
kUSDHC_Support4BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 0U),
kUSDHC_Support8BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 1U),
kUSDHC_SupportDDR50Flag = USDHC_HOST_CTRL_CAP_DDR50_SUPPORT_MASK,
kUSDHC_SupportSDR104Flag = USDHC_HOST_CTRL_CAP_SDR104_SUPPORT_MASK,
kUSDHC_SupportSDR50Flag = USDHC_HOST_CTRL_CAP_SDR50_SUPPORT_MASK }

```

*Host controller capabilities flag mask.*

- enum `_usdhc_wakeup_event` {
 

```

kUSDHC_WakeupEventOnCardInt = USDHC_PROT_CTRL_WECINT_MASK,
kUSDHC_WakeupEventOnCardInsert = USDHC_PROT_CTRL_WECINS_MASK,
kUSDHC_WakeupEventOnCardRemove = USDHC_PROT_CTRL_WECRM_MASK,
kUSDHC_WakeupEventsAll }

```

*Wakeup event mask.*

- enum `_usdhc_reset` {
 

```

kUSDHC_ResetAll = USDHC_SYS_CTRL_RSTA_MASK,
kUSDHC_ResetCommand = USDHC_SYS_CTRL_RSTC_MASK,
kUSDHC_ResetData = USDHC_SYS_CTRL_RSTD_MASK,
kUSDHC_ResetTuning = USDHC_SYS_CTRL_RSTT_MASK,
kUSDHC_ResetsAll }

```

*Reset type mask.*

- enum `_usdhc_transfer_flag` {
 

```

kUSDHC_EnableDmaFlag = USDHC_MIX_CTRL_DMAEN_MASK,
kUSDHC_CommandTypeSuspendFlag = (USDHC_CMD_XFR_TYP_CMDTYP(1U)),
kUSDHC_CommandTypeResumeFlag = (USDHC_CMD_XFR_TYP_CMDTYP(2U)),
kUSDHC_CommandTypeAbortFlag = (USDHC_CMD_XFR_TYP_CMDTYP(3U)),
kUSDHC_EnableBlockCountFlag = USDHC_MIX_CTRL_BCEN_MASK,
kUSDHC_EnableAutoCommand12Flag = USDHC_MIX_CTRL_AC12EN_MASK,
kUSDHC_DataReadFlag = USDHC_MIX_CTRL_DTDSEL_MASK,
kUSDHC_MultipleBlockFlag = USDHC_MIX_CTRL_MSBSEL_MASK,
kUSDHC_EnableAutoCommand23Flag = USDHC_MIX_CTRL_AC23EN_MASK,
kUSDHC_ResponseLength136Flag = USDHC_CMD_XFR_TYP_RSPTYP(1U),
kUSDHC_ResponseLength48Flag = USDHC_CMD_XFR_TYP_RSPTYP(2U),
kUSDHC_ResponseLength48BusyFlag = USDHC_CMD_XFR_TYP_RSPTYP(3U),
kUSDHC_EnableCrcCheckFlag = USDHC_CMD_XFR_TYP_CCCEN_MASK,
kUSDHC_EnableIndexCheckFlag = USDHC_CMD_XFR_TYP_CICEN_MASK,
kUSDHC_DataPresentFlag = USDHC_CMD_XFR_TYP_DPSEL_MASK }

```

*Transfer flag mask.*

- enum `_usdhc_present_status_flag` {

## Typical use case

```
kUSDHC_CommandInhibitFlag = USDHC_PRES_STATE_CIHB_MASK,  
kUSDHC_DataInhibitFlag = USDHC_PRES_STATE_CDIHB_MASK,  
kUSDHC_DataLineActiveFlag = USDHC_PRES_STATE_DLA_MASK,  
kUSDHC_SdClockStableFlag = USDHC_PRES_STATE_SDSTB_MASK,  
kUSDHC_WriteTransferActiveFlag = USDHC_PRES_STATE_WTA_MASK,  
kUSDHC_ReadTransferActiveFlag = USDHC_PRES_STATE_RTA_MASK,  
kUSDHC_BufferWriteEnableFlag = USDHC_PRES_STATE_BWEN_MASK,  
kUSDHC_BufferReadEnableFlag = USDHC_PRES_STATE_BREN_MASK,  
kUSDHC_ReTuningRequestFlag = USDHC_PRES_STATE_RTR_MASK,  
kUSDHC_DelaySettingFinishedFlag = USDHC_PRES_STATE_TSCD_MASK,  
kUSDHC_CardInsertedFlag = USDHC_PRES_STATE_CINST_MASK,  
kUSDHC_CommandLineLevelFlag = USDHC_PRES_STATE_CLSL_MASK,  
kUSDHC_Data0LineLevelFlag = (1U << USDHC_PRES_STATE_DLSSL_SHIFT),  
kUSDHC_Data1LineLevelFlag = (1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 1U)),  
kUSDHC_Data2LineLevelFlag = (1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 2U)),  
kUSDHC_Data3LineLevelFlag = (1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 3U)),  
kUSDHC_Data4LineLevelFlag = (1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 4U)),  
kUSDHC_Data5LineLevelFlag = (1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 5U)),  
kUSDHC_Data6LineLevelFlag = (1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 6U)),  
kUSDHC_Data7LineLevelFlag = (1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 7U)) }
```

*Present status flag mask.*

- enum `_usdhc_interrupt_status_flag` {

```

kUSDHC_CommandCompleteFlag = USDHC_INT_STATUS_CC_MASK,
kUSDHC_DataCompleteFlag = USDHC_INT_STATUS_TC_MASK,
kUSDHC_BlockGapEventFlag = USDHC_INT_STATUS_BGE_MASK,
kUSDHC_DmaCompleteFlag = USDHC_INT_STATUS_DINT_MASK,
kUSDHC_BufferWriteReadyFlag = USDHC_INT_STATUS_BWR_MASK,
kUSDHC_BufferReadReadyFlag = USDHC_INT_STATUS_BRR_MASK,
kUSDHC_CardInsertionFlag = USDHC_INT_STATUS_CINS_MASK,
kUSDHC_CardRemovalFlag = USDHC_INT_STATUS_CRM_MASK,
kUSDHC_CardInterruptFlag = USDHC_INT_STATUS_CINT_MASK,
kUSDHC_ReTuningEventFlag = USDHC_INT_STATUS_RTE_MASK,
kUSDHC_TuningPassFlag = USDHC_INT_STATUS_TP_MASK,
kUSDHC_TuningErrorFlag = USDHC_INT_STATUS_TNE_MASK,
kUSDHC_CommandTimeoutFlag = USDHC_INT_STATUS_CTOE_MASK,
kUSDHC_CommandCrcErrorFlag = USDHC_INT_STATUS_CCE_MASK,
kUSDHC_CommandEndBitErrorFlag = USDHC_INT_STATUS_CEBE_MASK,
kUSDHC_CommandIndexErrorFlag = USDHC_INT_STATUS_CIE_MASK,
kUSDHC_DataTimeoutFlag = USDHC_INT_STATUS_DTOE_MASK,
kUSDHC_DataCrcErrorFlag = USDHC_INT_STATUS_DCE_MASK,
kUSDHC_DataEndBitErrorFlag = USDHC_INT_STATUS_DEBE_MASK,
kUSDHC_AutoCommand12ErrorFlag = USDHC_INT_STATUS_AC12E_MASK,
kUSDHC_DmaErrorFlag = USDHC_INT_STATUS_DMAE_MASK,
kUSDHC_CommandErrorFlag,
kUSDHC_DataErrorFlag,
kUSDHC_ErrorFlag = (kUSDHC_CommandErrorFlag | kUSDHC_DataErrorFlag | kUSDHC_
DmaErrorFlag),
kUSDHC_DataFlag,
kUSDHC_CommandFlag = (kUSDHC_CommandErrorFlag | kUSDHC_CommandCompleteFlag),
kUSDHC_CardDetectFlag = (kUSDHC_CardInsertionFlag | kUSDHC_CardRemovalFlag) ,
kUSDHC_AllInterruptFlags }

```

*Interrupt status flag mask.*

- enum `_usdhc_auto_command12_error_status_flag` {

```

kUSDHC_AutoCommand12NotExecutedFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12-
NE_MASK,
kUSDHC_AutoCommand12TimeoutFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12TOE-
_MASK,
kUSDHC_AutoCommand12EndBitErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12-
EBE_MASK,
kUSDHC_AutoCommand12CrcErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12CE_-
_MASK,
kUSDHC_AutoCommand12IndexErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12IE-
_MASK,
kUSDHC_AutoCommand12NotIssuedFlag = USDHC_AUTOCMD12_ERR_STATUS_CNIBA-
C12E_MASK }

```

*Auto CMD12 error status flag mask.*

- enum `_usdhc_standard_tuning` {

## Typical use case

- ```
kUSDHC_ExecuteTuning = USDHC_AUTOCMD12_ERR_STATUS_EXECUTE_TUNING_M-
ASK,
kUSDHC_TuningSampleClockSel }
    standard tuning flag
```
- enum `_usdhc_adma_error_status_flag` {  
    `kUSDHC_AdmaLenghMismatchFlag` = `USDHC_ADMA_ERR_STATUS_ADMALME_MASK`,  
    `kUSDHC_AdmaDescriptorErrorFlag` = `USDHC_ADMA_ERR_STATUS_ADMADCE_MASK` }  
    *ADMA error status flag mask.*
  - enum `_usdhc_adma_error_state` {  
    `kUSDHC_AdmaErrorStateStopDma`,  
    `kUSDHC_AdmaErrorStateFetchDescriptor`,  
    `kUSDHC_AdmaErrorStateChangeAddress` = `0x02U`,  
    `kUSDHC_AdmaErrorStateTransferData`,  
    `kUSDHC_AdmaErrorStateInvalidLength` = `0x04U`,  
    `kUSDHC_AdmaErrorStateInvalidDescriptor` = `0x08U`,  
    `kUSDHC_AdmaErrorState` }  
    *ADMA error state.*
  - enum `_usdhc_force_event` {  
    `kUSDHC_ForceEventAutoCommand12NotExecuted` = `USDHC_FORCE_EVENT_FEVTAC12N-`  
    `E_MASK`,  
    `kUSDHC_ForceEventAutoCommand12Timeout` = `USDHC_FORCE_EVENT_FEVTAC12TOE_-`  
    `MASK`,  
    `kUSDHC_ForceEventAutoCommand12CrcError` = `USDHC_FORCE_EVENT_FEVTAC12CE_-`  
    `MASK`,  
    `kUSDHC_ForceEventEndBitError` = `USDHC_FORCE_EVENT_FEVTAC12EBE_MASK`,  
    `kUSDHC_ForceEventAutoCommand12IndexError` = `USDHC_FORCE_EVENT_FEVTAC12IE_-`  
    `MASK`,  
    `kUSDHC_ForceEventAutoCommand12NotIssued` = `USDHC_FORCE_EVENT_FEVTCNIBA-`  
    `C12E_MASK`,  
    `kUSDHC_ForceEventCommandTimeout` = `USDHC_FORCE_EVENT_FEVTCTOE_MASK`,  
    `kUSDHC_ForceEventCommandCrcError` = `USDHC_FORCE_EVENT_FEVTCCE_MASK`,  
    `kUSDHC_ForceEventCommandEndBitError` = `USDHC_FORCE_EVENT_FEVTCEBE_MASK`,  
    `kUSDHC_ForceEventCommandIndexError` = `USDHC_FORCE_EVENT_FEVTCIE_MASK`,  
    `kUSDHC_ForceEventDataTimeout` = `USDHC_FORCE_EVENT_FEVTDTOE_MASK`,  
    `kUSDHC_ForceEventDataCrcError` = `USDHC_FORCE_EVENT_FEVTDCE_MASK`,  
    `kUSDHC_ForceEventDataEndBitError` = `USDHC_FORCE_EVENT_FEVTDEBE_MASK`,  
    `kUSDHC_ForceEventAutoCommand12Error` = `USDHC_FORCE_EVENT_FEVTAC12E_MAS-`  
    `K`,  
    `kUSDHC_ForceEventCardInt` = `USDHC_FORCE_EVENT_FEVTCINT_MASK`,  
    `kUSDHC_ForceEventDmaError` = `USDHC_FORCE_EVENT_FEVTDMAE_MASK`,  
    `kUSDHC_ForceEventTuningError` = `USDHC_FORCE_EVENT_FEVTTNE_MASK`,  
    `kUSDHC_ForceEventsAll` }  
    *Force event mask.*
  - enum `usdhc_data_bus_width_t` {  
    `kUSDHC_DataBusWidth1Bit` = `0U`,  
    `kUSDHC_DataBusWidth4Bit` = `1U`,

- ```
kUSDHC_DataBusWidth8Bit = 2U }
```
- Data transfer width.*
- enum `usdhc_endian_mode_t` {  
`kUSDHC_EndianModeBig` = 0U,  
`kUSDHC_EndianModeHalfWordBig` = 1U,  
`kUSDHC_EndianModeLittle` = 2U }
- Endian mode.*
- enum `usdhc_dma_mode_t` {  
`kUSDHC_DmaModeSimple` = 0U,  
`kUSDHC_DmaModeAdma1` = 1U,  
`kUSDHC_DmaModeAdma2` = 2U,  
`kUSDHC_ExternalDMA` = 3U }
- DMA mode.*
- enum `_usdhc_sdio_control_flag` {  
`kUSDHC_StopAtBlockGapFlag` = USDHC\_PROT\_CTRL\_SABGREQ\_MASK,  
`kUSDHC_ReadWaitControlFlag` = USDHC\_PROT\_CTRL\_RWCTL\_MASK,  
`kUSDHC_InterruptAtBlockGapFlag` = USDHC\_PROT\_CTRL\_IABG\_MASK,  
`kUSDHC_ReadDoneNo8CLK` = USDHC\_PROT\_CTRL\_RD\_DONE\_NO\_8CLK\_MASK,  
`kUSDHC_ExactBlockNumberReadFlag` = USDHC\_PROT\_CTRL\_NON\_EXACT\_BLK\_RD\_M-  
ASK }
- SDIO control flag mask.*
- enum `usdhc_boot_mode_t` {  
`kUSDHC_BootModeNormal` = 0U,  
`kUSDHC_BootModeAlternative` = 1U }
- MMC card boot mode.*
- enum `usdhc_card_command_type_t` {  
`kCARD_CommandTypeNormal` = 0U,  
`kCARD_CommandTypeSuspend` = 1U,  
`kCARD_CommandTypeResume` = 2U,  
`kCARD_CommandTypeAbort` = 3U,  
`kCARD_CommandTypeEmpty` = 4U }
- The command type.*
- enum `usdhc_card_response_type_t` {  
`kCARD_ResponseTypeNone` = 0U,  
`kCARD_ResponseTypeR1` = 1U,  
`kCARD_ResponseTypeR1b` = 2U,  
`kCARD_ResponseTypeR2` = 3U,  
`kCARD_ResponseTypeR3` = 4U,  
`kCARD_ResponseTypeR4` = 5U,  
`kCARD_ResponseTypeR5` = 6U,  
`kCARD_ResponseTypeR5b` = 7U,  
`kCARD_ResponseTypeR6` = 8U,  
`kCARD_ResponseTypeR7` = 9U }
- The command response type.*
- enum `_usdhc_adma1_descriptor_flag` {

## Typical use case

```
kUSDHC_Adma1DescriptorValidFlag = (1U << 0U),
kUSDHC_Adma1DescriptorEndFlag = (1U << 1U),
kUSDHC_Adma1DescriptorInterruptFlag = (1U << 2U),
kUSDHC_Adma1DescriptorActivity1Flag = (1U << 4U),
kUSDHC_Adma1DescriptorActivity2Flag = (1U << 5U),
kUSDHC_Adma1DescriptorTypeNop = (kUSDHC_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeTransfer,
kUSDHC_Adma1DescriptorTypeLink,
kUSDHC_Adma1DescriptorTypeSetLength }
```

*The mask for the control/status field in ADMA1 descriptor.*

- enum `_usdhc_adma2_descriptor_flag` {  
kUSDHC\_Adma2DescriptorValidFlag = (1U << 0U),  
kUSDHC\_Adma2DescriptorEndFlag = (1U << 1U),  
kUSDHC\_Adma2DescriptorInterruptFlag = (1U << 2U),  
kUSDHC\_Adma2DescriptorActivity1Flag = (1U << 4U),  
kUSDHC\_Adma2DescriptorActivity2Flag = (1U << 5U),  
kUSDHC\_Adma2DescriptorTypeNop = (kUSDHC\_Adma2DescriptorValidFlag),  
kUSDHC\_Adma2DescriptorTypeReserved,  
kUSDHC\_Adma2DescriptorTypeTransfer,  
kUSDHC\_Adma2DescriptorTypeLink }

*ADMA1 descriptor control and status mask.*

- enum `_usdhc_adma_flag` {  
kUSDHC\_AdmaDescriptorSingleFlag,  
kUSDHC\_AdmaDescriptorMultipleFlag = 1U }

*ADMA descriptor configuration flag.*

- enum `usdhc_burst_len_t` {  
kUSDHC\_EnBurstLenForINCR = 0x01U,  
kUSDHC\_EnBurstLenForINCR4816 = 0x02U,  
kUSDHC\_EnBurstLenForINCR4816WRAP = 0x04U }

*dma transfer burst len config.*

- enum `_usdhc_transfer_data_type` {  
kUSDHC\_TransferDataNormal = 0U,  
kUSDHC\_TransferDataTuning = 1U,  
kUSDHC\_TransferDataBoot = 2U,  
kUSDHC\_TransferDataBootcontinuous = 3U }

*transfer data type definition.*

## Driver version

- #define `FSL_USDHC_DRIVER_VERSION` (MAKE\_VERSION(2U, 2U, 3U))  
*Driver version 2.2.3.*

## Initialization and deinitialization

- void `USDHC_Init` (USDHC\_Type \*base, const `usdhc_config_t` \*config)  
*USDHC module initialization function.*
- void `USDHC_Deinit` (USDHC\_Type \*base)

- *Deinitializes the USDHC.*
- bool **USDHC\_Reset** (USDHC\_Type \*base, uint32\_t mask, uint32\_t timeout)  
*Resets the USDHC.*

## DMA Control

- status\_t **USDHC\_SetAdmaTableConfig** (USDHC\_Type \*base, **usdhc\_adma\_config\_t** \*dmaConfig, **usdhc\_data\_t** \*dataConfig, uint32\_t flags)  
*Sets the DMA descriptor table configuration.*
- status\_t **USDHC\_SetInternalDmaConfig** (USDHC\_Type \*base, **usdhc\_adma\_config\_t** \*dmaConfig, const uint32\_t \*dataAddr, bool enAutoCmd23)  
*Internal DMA configuration.*
- status\_t **USDHC\_SetADMA2Descriptor** (uint32\_t \*admaTable, uint32\_t admaTableWords, const uint32\_t \*dataBufferAddr, uint32\_t dataBytes, uint32\_t flags)  
*Sets the ADMA2 descriptor table configuration.*
- status\_t **USDHC\_SetADMA1Descriptor** (uint32\_t \*admaTable, uint32\_t admaTableWords, const uint32\_t \*dataBufferAddr, uint32\_t dataBytes, uint32\_t flags)  
*Sets the ADMA1 descriptor table configuration.*
- static void **USDHC\_EnableInternalDMA** (USDHC\_Type \*base, bool enable)  
*enable internal DMA.*

## Interrupts

- static void **USDHC\_EnableInterruptStatus** (USDHC\_Type \*base, uint32\_t mask)  
*Enables the interrupt status.*
- static void **USDHC\_DisableInterruptStatus** (USDHC\_Type \*base, uint32\_t mask)  
*Disables the interrupt status.*
- static void **USDHC\_EnableInterruptSignal** (USDHC\_Type \*base, uint32\_t mask)  
*Enables the interrupt signal corresponding to the interrupt status flag.*
- static void **USDHC\_DisableInterruptSignal** (USDHC\_Type \*base, uint32\_t mask)  
*Disables the interrupt signal corresponding to the interrupt status flag.*

## Status

- static uint32\_t **USDHC\_GetInterruptStatusFlags** (USDHC\_Type \*base)  
*Gets the current interrupt status.*
- static void **USDHC\_ClearInterruptStatusFlags** (USDHC\_Type \*base, uint32\_t mask)  
*Clears a specified interrupt status.*
- static uint32\_t **USDHC\_GetAutoCommand12ErrorStatusFlags** (USDHC\_Type \*base)  
*Gets the status of auto command 12 error.*
- static uint32\_t **USDHC\_GetAdmaErrorStatusFlags** (USDHC\_Type \*base)  
*Gets the status of the ADMA error.*
- static uint32\_t **USDHC\_GetPresentStatusFlags** (USDHC\_Type \*base)  
*Gets a present status.*

## Bus Operations

- void **USDHC\_GetCapability** (USDHC\_Type \*base, **usdhc\_capability\_t** \*capability)  
*Gets the capability information.*
- static void **USDHC\_ForceClockOn** (USDHC\_Type \*base, bool enable)

## Typical use case

- force the card clock on.*
- uint32\_t [USDHC\\_SetSdClock](#) (USDHC\_Type \*base, uint32\_t srcClock\_Hz, uint32\_t busClock\_Hz)  
*Sets the SD bus clock frequency.*
- bool [USDHC\\_SetCardActive](#) (USDHC\_Type \*base, uint32\_t timeout)  
*Sends 80 clocks to the card to set it to the active state.*
- static void [USDHC\\_AssertHardwareReset](#) (USDHC\_Type \*base, bool high)  
*trigger a hardware reset.*
- static void [USDHC\\_SetDataBusWidth](#) (USDHC\_Type \*base, [usdhc\\_data\\_bus\\_width\\_t](#) width)  
*Sets the data transfer width.*
- static void [USDHC\\_WriteData](#) (USDHC\_Type \*base, uint32\_t data)  
*Fills the data port.*
- static uint32\_t [USDHC\\_ReadData](#) (USDHC\_Type \*base)  
*Retrieves the data from the data port.*
- void [USDHC\\_SendCommand](#) (USDHC\_Type \*base, [usdhc\\_command\\_t](#) \*command)  
*send command function*
- static void [USDHC\\_EnableWakeupEvent](#) (USDHC\_Type \*base, uint32\_t mask, bool enable)  
*Enables or disables a wakeup event in low-power mode.*
- static void [USDHC\\_CardDetectByData3](#) (USDHC\_Type \*base, bool enable)  
*detect card insert status.*
- static bool [USDHC\\_DetectCardInsert](#) (USDHC\_Type \*base)  
*detect card insert status.*
- static void [USDHC\\_EnableSdioControl](#) (USDHC\_Type \*base, uint32\_t mask, bool enable)  
*Enables or disables the SDIO card control.*
- static void [USDHC\\_SetContinueRequest](#) (USDHC\_Type \*base)  
*Restarts a transaction which has stopped at the block GAP for the SDIO card.*
- static void [USDHC\\_RequestStopAtBlockGap](#) (USDHC\_Type \*base, bool enable)  
*Request stop at block gap function.*
- void [USDHC\\_SetMmcBootConfig](#) (USDHC\_Type \*base, const [usdhc\\_boot\\_config\\_t](#) \*config)  
*Configures the MMC boot feature.*
- static void [USDHC\\_EnableMmcBoot](#) (USDHC\_Type \*base, bool enable)  
*Enables or disables the mmc boot mode.*
- static void [USDHC\\_SetForceEvent](#) (USDHC\_Type \*base, uint32\_t mask)  
*Forces generating events according to the given mask.*
- static void [USDHC\\_SelectVoltage](#) (USDHC\_Type \*base, bool en18v)  
*select the usdhc output voltage*
- static bool [USDHC\\_RequestTuningForSDR50](#) (USDHC\_Type \*base)  
*check the SDR50 mode request tuning bit When this bit set, user should call USDHC\_StandardTuning function*
- static bool [USDHC\\_RequestReTuning](#) (USDHC\_Type \*base)  
*check the request re-tuning bit When this bit is set, user should do manual tuning or standard tuning function*
- static void [USDHC\\_EnableAutoTuning](#) (USDHC\_Type \*base, bool enable)  
*the SDR104 mode auto tuning enable and disable This function should call after tuning function execute pass, auto tuning will handle by hardware*
- static void [USDHC\\_SetRetuningTimer](#) (USDHC\_Type \*base, uint32\_t counter)  
*the config the re-tuning timer for mode 1 and mode 3 This timer is used for standard tuning auto re-tuning,*
- void [USDHC\\_EnableAutoTuningForCmdAndData](#) (USDHC\_Type \*base)  
*the auto tuning enable for CMD/DATA line*
- void [USDHC\\_EnableManualTuning](#) (USDHC\_Type \*base, bool enable)  
*manual tuning trigger or abort User should handle the tuning cmd and find the boundary of the delay then*



calculate a average value which will be config to the `CLK_TUNE_CTRL_STATUS` This function should called before `USDHC_AdjustDelayForSDR104` function

- `status_t USDHC_AdjustDelayForManualTuning` (`USDHC_Type *base`, `uint32_t delay`)  
the SDR104 mode delay setting adjust This function should called after `USDHC_ManualTuningForSDR104`
- `void USDHC_EnableStandardTuning` (`USDHC_Type *base`, `uint32_t tuningStartTap`, `uint32_t step`, `bool enable`)  
the enable standard tuning function The standard tuning window and tuning counter use the default config tuning cmd is send by the software, user need to check the tuning result can be used for SDR50,SDR104,H-S200 mode tuning
- `static uint32_t USDHC_GetExecuteStdTuningStatus` (`USDHC_Type *base`)  
Get execute std tuning status.
- `static uint32_t USDHC_CheckStdTuningResult` (`USDHC_Type *base`)  
check std tuning result
- `static uint32_t USDHC_CheckTuningError` (`USDHC_Type *base`)  
check tuning error
- `void USDHC_EnableDDRMMode` (`USDHC_Type *base`, `bool enable`, `uint32_t nibblePos`)  
the enable/disable DDR mode

### Transactional

the enable/disable HS400 mode

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable/disable</i>	flag

- `status_t USDHC_TransferBlocking` (`USDHC_Type *base`, `usdhc_adma_config_t *dmaConfig`, `usdhc_transfer_t *transfer`)  
Transfers the command/data using a blocking method.
- `void USDHC_TransferCreateHandle` (`USDHC_Type *base`, `usdhc_handle_t *handle`, `const usdhc_transfer_callback_t *callback`, `void *userData`)  
Creates the USDHC handle.
- `status_t USDHC_TransferNonBlocking` (`USDHC_Type *base`, `usdhc_handle_t *handle`, `usdhc_adma_config_t *dmaConfig`, `usdhc_transfer_t *transfer`)  
Transfers the command/data using an interrupt and an asynchronous method.
- `void USDHC_TransferHandleIRQ` (`USDHC_Type *base`, `usdhc_handle_t *handle`)  
IRQ handler for the USDHC.

## 45.3 Data Structure Documentation

### 45.3.1 struct usdhc\_adma2\_descriptor\_t

#### Data Fields

- `uint32_t attribute`  
The control and status field.
- `const uint32_t * address`

## Data Structure Documentation

*The address field.*

### 45.3.2 struct usdhc\_capability\_t

Defines a structure to save the capability information of USDHC.

#### Data Fields

- uint32\_t [sdVersion](#)  
*support SD card/sdio version*
- uint32\_t [mmcVersion](#)  
*support emmc card version*
- uint32\_t [maxBlockLength](#)  
*Maximum block length united as byte.*
- uint32\_t [maxBlockCount](#)  
*Maximum block count can be set one time.*
- uint32\_t [flags](#)  
*Capability flags to indicate the support information([\\_usdhc\\_capability\\_flag](#))*

### 45.3.3 struct usdhc\_boot\_config\_t

#### Data Fields

- uint32\_t [ackTimeoutCount](#)  
*Timeout value for the boot ACK.*
- [usdhc\\_boot\\_mode\\_t](#) [bootMode](#)  
*Boot mode selection.*
- uint32\_t [blockCount](#)  
*Stop at block gap value of automatic mode.*
- size\_t [blockSize](#)  
*Block size.*
- bool [enableBootAck](#)  
*Enable or disable boot ACK.*
- bool [enableAutoStopAtBlockGap](#)  
*Enable or disable auto stop at block gap function in boot period.*

#### 45.3.3.0.0.59 Field Documentation

##### 45.3.3.0.0.59.1 uint32\_t usdhc\_boot\_config\_t::ackTimeoutCount

The available range is 0 ~ 15.

**45.3.3.0.0.59.2** `usdhc_boot_mode_t usdhc_boot_config_t::bootMode`

**45.3.3.0.0.59.3** `uint32_t usdhc_boot_config_t::blockCount`

Available range is 0 ~ 65535.

## 45.3.4 struct `usdhc_config_t`

### Data Fields

- `uint32_t dataTimeout`  
*Data timeout value.*
- `usdhc_endian_mode_t endianMode`  
*Endian mode.*
- `uint8_t readWatermarkLevel`  
*Watermark level for DMA read operation.*
- `uint8_t writeWatermarkLevel`  
*Watermark level for DMA write operation.*
- `uint8_t readBurstLen`  
*Read burst len.*
- `uint8_t writeBurstLen`  
*Write burst len.*

### 45.3.4.0.0.60 Field Documentation

**45.3.4.0.0.60.1** `uint8_t usdhc_config_t::readWatermarkLevel`

Available range is 1 ~ 128.

**45.3.4.0.0.60.2** `uint8_t usdhc_config_t::writeWatermarkLevel`

Available range is 1 ~ 128.

## 45.3.5 struct `usdhc_data_t`

Defines a structure to contain data-related attribute. 'enableIgnoreError' is used for the case that upper card driver want to ignore the error event to read/write all the data not to stop read/write immediately when error event happen for example bus testing procedure for MMC card.

### Data Fields

- `bool enableAutoCommand12`  
*Enable auto CMD12.*
- `bool enableAutoCommand23`  
*Enable auto CMD23.*
- `bool enableIgnoreError`

## Data Structure Documentation

- *Enable to ignore error event to read/write all the data.*
- uint8\_t **dataType**  
*this is used to distinguish the normal/tuning/boot data*
- size\_t **blockSize**  
*Block size.*
- uint32\_t **blockCount**  
*Block count.*
- uint32\_t \* **rxData**  
*Buffer to save data read.*
- const uint32\_t \* **txData**  
*Data buffer to write.*

### 45.3.6 struct usdhc\_command\_t

Define card command-related attribute.

#### Data Fields

- uint32\_t **index**  
*Command index.*
- uint32\_t **argument**  
*Command argument.*
- usdhc\_card\_command\_type\_t type  
*Command type.*
- usdhc\_card\_response\_type\_t responseType  
*Command response type.*
- uint32\_t **response** [4U]  
*Response for this command.*
- uint32\_t **responseErrorFlags**  
*response error flag, the flag which need to check the command reponse*
- uint32\_t **flags**  
*Cmd flags.*

### 45.3.7 struct usdhc\_adma\_config\_t

#### Data Fields

- usdhc\_dma\_mode\_t dmaMode  
*DMA mode.*
- usdhc\_burst\_len\_t burstLen  
*burst len config*
- uint32\_t \* **admaTable**  
*ADMA table address, can't be null if transfer way is ADMA1/ADMA2.*
- uint32\_t **admaTableWords**  
*ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.*

### 45.3.8 struct usdhc\_transfer\_t

#### Data Fields

- [usdhc\\_data\\_t](#) \* data  
*Data to transfer.*
- [usdhc\\_command\\_t](#) \* command  
*Command to send.*

### 45.3.9 struct usdhc\_transfer\_callback\_t

#### Data Fields

- void(\* [CardInserted](#) )(USDHC\_Type \*base, void \*userData)  
*Card inserted occurs when DAT3/CD pin is for card detect.*
- void(\* [CardRemoved](#) )(USDHC\_Type \*base, void \*userData)  
*Card removed occurs.*
- void(\* [SdioInterrupt](#) )(USDHC\_Type \*base, void \*userData)  
*SDIO card interrupt occurs.*
- void(\* [BlockGap](#) )(USDHC\_Type \*base, void \*userData)  
*stopped at block gap event*
- void(\* [TransferComplete](#) )(USDHC\_Type \*base, usdhc\_handle\_t \*handle, status\_t status, void \*userData)  
*Transfer complete callback.*
- void(\* [ReTuning](#) )(USDHC\_Type \*base, void \*userData)  
*handle the re-tuning*

### 45.3.10 struct \_usdhc\_handle

USDHC handle typedef.

Defines the structure to save the USDHC state information and callback function. The detailed interrupt status when sending a command or transferring data can be obtained from the interruptFlags field by using the mask defined in usdhc\_interrupt\_flag\_t.

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

#### Data Fields

- [usdhc\\_data\\_t](#) \*volatile data  
*Data to transfer.*
- [usdhc\\_command\\_t](#) \*volatile command  
*Command to send.*

## Enumeration Type Documentation

- volatile uint32\_t [interruptFlags](#)  
*Interrupt flags of last transaction.*
- volatile uint32\_t [transferredWords](#)  
*Words transferred by DATAPORT way.*
- [usdhc\\_transfer\\_callback\\_t](#) [callback](#)  
*Callback function.*
- void \* [userData](#)  
*Parameter for transfer complete callback.*

### 45.3.11 struct usdhc\_host\_t

#### Data Fields

- USDHC\_Type \* [base](#)  
*USDHC peripheral base address.*
- uint32\_t [sourceClock\\_Hz](#)  
*USDHC source clock frequency united in Hz.*
- [usdhc\\_config\\_t](#) [config](#)  
*USDHC configuration.*
- [usdhc\\_capability\\_t](#) [capability](#)  
*USDHC capability information.*
- [usdhc\\_transfer\\_function\\_t](#) [transfer](#)  
*USDHC transfer function.*

## 45.4 Macro Definition Documentation

### 45.4.1 #define FSL\_USDHC\_DRIVER\_VERSION (MAKE\_VERSION(2U, 2U, 3U))

## 45.5 Typedef Documentation

### 45.5.1 typedef uint32\_t usdhc\_adma1\_descriptor\_t

### 45.5.2 typedef status\_t(\* usdhc\_transfer\_function\_t)(USDHC\_Type \*base, usdhc\_transfer\_t \*content)

## 45.6 Enumeration Type Documentation

### 45.6.1 enum \_usdhc\_status

Enumerator

- kStatus\_USDHC\_BusyTransferring* Transfer is on-going.
- kStatus\_USDHC\_PrepareAdmaDescriptorFailed* Set DMA descriptor failed.
- kStatus\_USDHC\_SendCommandFailed* Send command failed.
- kStatus\_USDHC\_TransferDataFailed* Transfer data failed.
- kStatus\_USDHC\_DMADDataAddrNotAlign* data address not align
- kStatus\_USDHC\_ReTuningRequest* re-tuning request

*kStatus\_USDHC\_TuningError* tuning error  
*kStatus\_USDHC\_NotSupport* not support

### 45.6.2 enum \_usdhc\_capability\_flag

Enumerator

*kUSDHC\_SupportAdmaFlag* Support ADMA.  
*kUSDHC\_SupportHighSpeedFlag* Support high-speed.  
*kUSDHC\_SupportDmaFlag* Support DMA.  
*kUSDHC\_SupportSuspendResumeFlag* Support suspend/resume.  
*kUSDHC\_SupportV330Flag* Support voltage 3.3V.  
*kUSDHC\_SupportV300Flag* Support voltage 3.0V.  
*kUSDHC\_SupportV180Flag* Support voltage 1.8V.  
*kUSDHC\_Support4BitFlag* Support 4 bit mode.  
*kUSDHC\_Support8BitFlag* Support 8 bit mode.  
*kUSDHC\_SupportDDR50Flag* support DDR50 mode  
*kUSDHC\_SupportSDR104Flag* support SDR104 mode  
*kUSDHC\_SupportSDR50Flag* support SDR50 mode

### 45.6.3 enum \_usdhc\_wakeup\_event

Enumerator

*kUSDHC\_WakeupEventOnCardInt* Wakeup on card interrupt.  
*kUSDHC\_WakeupEventOnCardInsert* Wakeup on card insertion.  
*kUSDHC\_WakeupEventOnCardRemove* Wakeup on card removal.  
*kUSDHC\_WakeupEventsAll* All wakeup events.

### 45.6.4 enum \_usdhc\_reset

Enumerator

*kUSDHC\_ResetAll* Reset all except card detection.  
*kUSDHC\_ResetCommand* Reset command line.  
*kUSDHC\_ResetData* Reset data line.  
*kUSDHC\_ResetTuning* reset tuning circuit  
*kUSDHC\_ResetsAll* All reset types.

## Enumeration Type Documentation

### 45.6.5 enum \_usdhc\_transfer\_flag

Enumerator

*kUSDHC\_EnableDmaFlag* Enable DMA.  
*kUSDHC\_CommandTypeSuspendFlag* Suspend command.  
*kUSDHC\_CommandTypeResumeFlag* Resume command.  
*kUSDHC\_CommandTypeAbortFlag* Abort command.  
*kUSDHC\_EnableBlockCountFlag* Enable block count.  
*kUSDHC\_EnableAutoCommand12Flag* Enable auto CMD12.  
*kUSDHC\_DataReadFlag* Enable data read.  
*kUSDHC\_MultipleBlockFlag* Multiple block data read/write.  
*kUSDHC\_EnableAutoCommand23Flag* Enable auto CMD23.  
*kUSDHC\_ResponseLength136Flag* 136 bit response length  
*kUSDHC\_ResponseLength48Flag* 48 bit response length  
*kUSDHC\_ResponseLength48BusyFlag* 48 bit response length with busy status  
*kUSDHC\_EnableCrcCheckFlag* Enable CRC check.  
*kUSDHC\_EnableIndexCheckFlag* Enable index check.  
*kUSDHC\_DataPresentFlag* Data present flag.

### 45.6.6 enum \_usdhc\_present\_status\_flag

Enumerator

*kUSDHC\_CommandInhibitFlag* Command inhibit.  
*kUSDHC\_DataInhibitFlag* Data inhibit.  
*kUSDHC\_DataLineActiveFlag* Data line active.  
*kUSDHC\_SdClockStableFlag* SD bus clock stable.  
*kUSDHC\_WriteTransferActiveFlag* Write transfer active.  
*kUSDHC\_ReadTransferActiveFlag* Read transfer active.  
*kUSDHC\_BufferWriteEnableFlag* Buffer write enable.  
*kUSDHC\_BufferReadEnableFlag* Buffer read enable.  
*kUSDHC\_ReTuningRequestFlag* re-tuning request flag ,only used for SDR104 mode  
*kUSDHC\_DelaySettingFinishedFlag* delay setting finished flag  
*kUSDHC\_CardInsertedFlag* Card inserted.  
*kUSDHC\_CommandLineLevelFlag* Command line signal level.  
*kUSDHC\_Data0LineLevelFlag* Data0 line signal level.  
*kUSDHC\_Data1LineLevelFlag* Data1 line signal level.  
*kUSDHC\_Data2LineLevelFlag* Data2 line signal level.  
*kUSDHC\_Data3LineLevelFlag* Data3 line signal level.  
*kUSDHC\_Data4LineLevelFlag* Data4 line signal level.  
*kUSDHC\_Data5LineLevelFlag* Data5 line signal level.  
*kUSDHC\_Data6LineLevelFlag* Data6 line signal level.  
*kUSDHC\_Data7LineLevelFlag* Data7 line signal level.



### 45.6.7 enum \_usdhc\_interrupt\_status\_flag

Enumerator

*kUSDHC\_CommandCompleteFlag* Command complete.  
*kUSDHC\_DataCompleteFlag* Data complete.  
*kUSDHC\_BlockGapEventFlag* Block gap event.  
*kUSDHC\_DmaCompleteFlag* DMA interrupt.  
*kUSDHC\_BufferWriteReadyFlag* Buffer write ready.  
*kUSDHC\_BufferReadReadyFlag* Buffer read ready.  
*kUSDHC\_CardInsertionFlag* Card inserted.  
*kUSDHC\_CardRemovalFlag* Card removed.  
*kUSDHC\_CardInterruptFlag* Card interrupt.  
*kUSDHC\_ReTuningEventFlag* Re-Tuning event, only for SD3.0 SDR104 mode.  
*kUSDHC\_TuningPassFlag* SDR104 mode tuning pass flag.  
*kUSDHC\_TuningErrorFlag* SDR104 tuning error flag.  
*kUSDHC\_CommandTimeoutFlag* Command timeout error.  
*kUSDHC\_CommandCrcErrorFlag* Command CRC error.  
*kUSDHC\_CommandEndBitErrorFlag* Command end bit error.  
*kUSDHC\_CommandIndexErrorFlag* Command index error.  
*kUSDHC\_DataTimeoutFlag* Data timeout error.  
*kUSDHC\_DataCrcErrorFlag* Data CRC error.  
*kUSDHC\_DataEndBitErrorFlag* Data end bit error.  
*kUSDHC\_AutoCommand12ErrorFlag* Auto CMD12 error.  
*kUSDHC\_DmaErrorFlag* DMA error.  
*kUSDHC\_CommandErrorFlag* Command error.  
*kUSDHC\_DataErrorFlag* Data error.  
*kUSDHC\_ErrorFlag* All error.  
*kUSDHC\_DataFlag* Data interrupts.  
*kUSDHC\_CommandFlag* Command interrupts.  
*kUSDHC\_CardDetectFlag* Card detection interrupts.  
*kUSDHC\_AllInterruptFlags* All flags mask.

### 45.6.8 enum \_usdhc\_auto\_command12\_error\_status\_flag

Enumerator

*kUSDHC\_AutoCommand12NotExecutedFlag* Not executed error.  
*kUSDHC\_AutoCommand12TimeoutFlag* Timeout error.  
*kUSDHC\_AutoCommand12EndBitErrorFlag* End bit error.  
*kUSDHC\_AutoCommand12CrcErrorFlag* CRC error.  
*kUSDHC\_AutoCommand12IndexErrorFlag* Index error.  
*kUSDHC\_AutoCommand12NotIssuedFlag* Not issued error.

## Enumeration Type Documentation

### 45.6.9 enum \_usdhc\_standard\_tuning

Enumerator

*kUSDHC\_ExecuteTuning* used to start tuning procedure

*kUSDHC\_TuningSampleClockSel* when std\_tuning\_en bit is set, this bit is used select sampling clock

### 45.6.10 enum \_usdhc\_adma\_error\_status\_flag

Enumerator

*kUSDHC\_AdmaLenghMismatchFlag* Length mismatch error.

*kUSDHC\_AdmaDescriptorErrorFlag* Descriptor error.

### 45.6.11 enum \_usdhc\_adma\_error\_state

This state is the detail state when ADMA error has occurred.

Enumerator

*kUSDHC\_AdmaErrorStateStopDma* Stop DMA, previous location set in the ADMA system address is error address.

*kUSDHC\_AdmaErrorStateFetchDescriptor* Fetch descriptor, current location set in the ADMA system address is error address.

*kUSDHC\_AdmaErrorStateChangeAddress* Change address, no DMA error is occurred.

*kUSDHC\_AdmaErrorStateTransferData* Transfer data, previous location set in the ADMA system address is error address.

*kUSDHC\_AdmaErrorStateInvalidLength* Invalid length in ADMA descriptor.

*kUSDHC\_AdmaErrorStateInvalidDescriptor* Invalid descriptor fetched by ADMA.

*kUSDHC\_AdmaErrorState* ADMA error state.

### 45.6.12 enum \_usdhc\_force\_event

Enumerator

*kUSDHC\_ForceEventAutoCommand12NotExecuted* Auto CMD12 not executed error.

*kUSDHC\_ForceEventAutoCommand12Timeout* Auto CMD12 timeout error.

*kUSDHC\_ForceEventAutoCommand12CrcError* Auto CMD12 CRC error.

*kUSDHC\_ForceEventEndBitError* Auto CMD12 end bit error.

*kUSDHC\_ForceEventAutoCommand12IndexError* Auto CMD12 index error.

*kUSDHC\_ForceEventAutoCommand12NotIssued* Auto CMD12 not issued error.

*kUSDHC\_ForceEventCommandTimeout* Command timeout error.  
*kUSDHC\_ForceEventCommandCrcError* Command CRC error.  
*kUSDHC\_ForceEventCommandEndBitError* Command end bit error.  
*kUSDHC\_ForceEventCommandIndexError* Command index error.  
*kUSDHC\_ForceEventDataTimeout* Data timeout error.  
*kUSDHC\_ForceEventDataCrcError* Data CRC error.  
*kUSDHC\_ForceEventDataEndBitError* Data end bit error.  
*kUSDHC\_ForceEventAutoCommand12Error* Auto CMD12 error.  
*kUSDHC\_ForceEventCardInt* Card interrupt.  
*kUSDHC\_ForceEventDmaError* Dma error.  
*kUSDHC\_ForceEventTuningError* Tuning error.  
*kUSDHC\_ForceEventsAll* All force event flags mask.

### 45.6.13 enum usdhc\_data\_bus\_width\_t

Enumerator

*kUSDHC\_DataBusWidth1Bit* 1-bit mode  
*kUSDHC\_DataBusWidth4Bit* 4-bit mode  
*kUSDHC\_DataBusWidth8Bit* 8-bit mode

### 45.6.14 enum usdhc\_endian\_mode\_t

Enumerator

*kUSDHC\_EndianModeBig* Big endian mode.  
*kUSDHC\_EndianModeHalfWordBig* Half word big endian mode.  
*kUSDHC\_EndianModeLittle* Little endian mode.

### 45.6.15 enum usdhc\_dma\_mode\_t

Enumerator

*kUSDHC\_DmaModeSimple* external DMA  
*kUSDHC\_DmaModeAdma1* ADMA1 is selected.  
*kUSDHC\_DmaModeAdma2* ADMA2 is selected.  
*kUSDHC\_ExternalDMA* external dma mode select

## Enumeration Type Documentation

### 45.6.16 enum \_usdhc\_sdio\_control\_flag

Enumerator

*kUSDHC\_StopAtBlockGapFlag* Stop at block gap.  
*kUSDHC\_ReadWaitControlFlag* Read wait control.  
*kUSDHC\_InterruptAtBlockGapFlag* Interrupt at block gap.  
*kUSDHC\_ReadDoneNo8CLK* read done without 8 clk for block gap  
*kUSDHC\_ExactBlockNumberReadFlag* Exact block number read.

### 45.6.17 enum usdhc\_boot\_mode\_t

Enumerator

*kUSDHC\_BootModeNormal* Normal boot.  
*kUSDHC\_BootModeAlternative* Alternative boot.

### 45.6.18 enum usdhc\_card\_command\_type\_t

Enumerator

*kCARD\_CommandTypeNormal* Normal command.  
*kCARD\_CommandTypeSuspend* Suspend command.  
*kCARD\_CommandTypeResume* Resume command.  
*kCARD\_CommandTypeAbort* Abort command.  
*kCARD\_CommandTypeEmpty* Empty command.

### 45.6.19 enum usdhc\_card\_response\_type\_t

Define the command response type from card to host controller.

Enumerator

*kCARD\_ResponseTypeNone* Response type: none.  
*kCARD\_ResponseTypeR1* Response type: R1.  
*kCARD\_ResponseTypeR1b* Response type: R1b.  
*kCARD\_ResponseTypeR2* Response type: R2.  
*kCARD\_ResponseTypeR3* Response type: R3.  
*kCARD\_ResponseTypeR4* Response type: R4.  
*kCARD\_ResponseTypeR5* Response type: R5.  
*kCARD\_ResponseTypeR5b* Response type: R5b.  
*kCARD\_ResponseTypeR6* Response type: R6.  
*kCARD\_ResponseTypeR7* Response type: R7.

### 45.6.20 enum \_usdhc\_adma1\_descriptor\_flag

Enumerator

*kUSDHC\_Adma1DescriptorValidFlag* Valid flag.  
*kUSDHC\_Adma1DescriptorEndFlag* End flag.  
*kUSDHC\_Adma1DescriptorInterruptFlag* Interrupt flag.  
*kUSDHC\_Adma1DescriptorActivity1Flag* Activity 1 flag.  
*kUSDHC\_Adma1DescriptorActivity2Flag* Activity 2 flag.  
*kUSDHC\_Adma1DescriptorTypeNop* No operation.  
*kUSDHC\_Adma1DescriptorTypeTransfer* Transfer data.  
*kUSDHC\_Adma1DescriptorTypeLink* Link descriptor.  
*kUSDHC\_Adma1DescriptorTypeSetLength* Set data length.

### 45.6.21 enum \_usdhc\_adma2\_descriptor\_flag

Enumerator

*kUSDHC\_Adma2DescriptorValidFlag* Valid flag.  
*kUSDHC\_Adma2DescriptorEndFlag* End flag.  
*kUSDHC\_Adma2DescriptorInterruptFlag* Interrupt flag.  
*kUSDHC\_Adma2DescriptorActivity1Flag* Activity 1 mask.  
*kUSDHC\_Adma2DescriptorActivity2Flag* Activity 2 mask.  
*kUSDHC\_Adma2DescriptorTypeNop* No operation.  
*kUSDHC\_Adma2DescriptorTypeReserved* Reserved.  
*kUSDHC\_Adma2DescriptorTypeTransfer* Transfer type.  
*kUSDHC\_Adma2DescriptorTypeLink* Link type.

### 45.6.22 enum \_usdhc\_adma\_flag

Enumerator

*kUSDHC\_AdmaDescriptorSingleFlag* try to finish the transfer in a single ADMA descriptor, if transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer  
*kUSDHC\_AdmaDescriptorMultipleFlag* create multiple ADMA descriptor within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combine with stop at block gap feature

## Function Documentation

### 45.6.23 enum usdhc\_burst\_len\_t

Enumerator

*kUSDHC\_EnBurstLenForINCR* enable burst len for INCR  
*kUSDHC\_EnBurstLenForINCR4816* enable burst len for INCR4/INCR8/INCR16  
*kUSDHC\_EnBurstLenForINCR4816WRAP* enable burst len for INCR4/8/16 WRAP

### 45.6.24 enum \_usdhc\_transfer\_data\_type

Enumerator

*kUSDHC\_TransferDataNormal* transfer normal read/write data  
*kUSDHC\_TransferDataTuning* transfer tuning data  
*kUSDHC\_TransferDataBoot* transfer boot data  
*kUSDHC\_TransferDataBootcontinous* transfer boot data continous

## 45.7 Function Documentation

### 45.7.1 void USDHC\_Init ( USDHC\_Type \* *base*, const usdhc\_config\_t \* *config* )

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;  
config.cardDetectDat3 = false;  
config.endianMode = kUSDHC_EndianModeLittle;  
config.dmaMode = kUSDHC_DmaModeAdma2;  
config.readWatermarkLevel = 128U;  
config.writeWatermarkLevel = 128U;  
USDHC_Init(USDHC, &config);
```

Parameters

<i>base</i>	USDHC peripheral base address.
<i>config</i>	USDHC configuration information.

Return values

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

### 45.7.2 void USDHC\_Deinit ( USDHC\_Type \* *base* )

Parameters

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

### 45.7.3 `bool USDHC_Reset ( USDHC_Type * base, uint32_t mask, uint32_t timeout )`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The reset type mask( <code>_usdhc_reset</code> ).
<i>timeout</i>	Timeout for reset.

Return values

<i>true</i>	Reset successfully.
<i>false</i>	Reset failed.

### 45.7.4 `status_t USDHC_SetAdmaTableConfig ( USDHC_Type * base, usdhc_adma_config_t * dmaConfig, usdhc_data_t * dataConfig, uint32_t flags )`

A high level DMA descriptor configuration function.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>adma</i>	configuration
<i>data</i>	Data descriptor
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please reference <code>_usdhc_adma_flag</code>

Return values

## Function Documentation

<i>kStatus_OutOfRange</i>	ADMA descriptor table length isn't enough to describe data.
<i>kStatus_Success</i>	Operate successfully.

### 45.7.5 **status\_t** USDHC\_SetInternalDmaConfig ( **USDHC\_Type** \* *base*, **usdhc\_adma\_config\_t** \* *dmaConfig*, **const uint32\_t** \* *dataAddr*, **bool** *enAutoCmd23* )

This function is used to config the USDHC DMA related registers.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>adma</i>	configuration
<i>dataAddr</i>	transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
<i>enAutoCmd23</i>	flag to indicate Auto CMD23 is enable or not, a simple DMA parameter,if ADMA is used, leave it to false.

Return values

<i>kStatus_OutOfRange</i>	ADMA descriptor table length isn't enough to describe data.
<i>kStatus_Success</i>	Operate successfully.

### 45.7.6 **status\_t** USDHC\_SetADMA2Descriptor ( **uint32\_t** \* *admaTable*, **uint32\_t** *admaTableWords*, **const uint32\_t** \* *dataBufferAddr*, **uint32\_t** *dataBytes*, **uint32\_t** *flags* )

Parameters

<i>admaTable</i>	Adma table address.
<i>admaTableWords</i>	Adma table length.
<i>dataBufferAddr</i>	Data buffer address.
<i>dataBytes</i>	Data Data length.
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please reference <code>_usdhc_adma_flag</code> .



Return values

<i>kStatus_OutOfRange</i>	ADMA descriptor table length isn't enough to describe data.
<i>kStatus_Success</i>	Operate successfully.

**45.7.7** `status_t USDHC_SetADMA1Descriptor ( uint32_t * admaTable, uint32_t admaTableWords, const uint32_t * dataBufferAddr, uint32_t dataBytes, uint32_t flags )`

Parameters

<i>admaTable</i>	Adma table address.
<i>admaTableWords</i>	Adma table length.
<i>dataBufferAddr</i>	Data buffer address.
<i>dataBytes</i>	Data length.
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please reference <code>_usdhc_adma_flag</code> .

Return values

<i>kStatus_OutOfRange</i>	ADMA descriptor table length isn't enough to describe data.
<i>kStatus_Success</i>	Operate successfully.

**45.7.8** `static void USDHC_EnableInternalDMA ( USDHC_Type * base, bool enable ) [inline], [static]`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable or disable flag

**45.7.9** `static void USDHC_EnableInterruptStatus ( USDHC_Type * base, uint32_t mask ) [inline], [static]`

## Function Documentation

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	Interrupt status flags mask(_usdhc_interrupt_status_flag).

**45.7.10 static void USDHC\_DisableInterruptStatus ( USDHC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

**45.7.11 static void USDHC\_EnableInterruptSignal ( USDHC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

**45.7.12 static void USDHC\_DisableInterruptSignal ( USDHC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

**45.7.13 static uint32\_t USDHC\_GetInterruptStatusFlags ( USDHC\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

Current interrupt status flags mask(\_usdhc\_interrupt\_status\_flag).

**45.7.14 static void USDHC\_ClearInterruptStatusFlags ( USDHC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

write 1 clears

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

**45.7.15 static uint32\_t USDHC\_GetAutoCommand12ErrorStatusFlags ( USDHC\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

Auto command 12 error status flags mask(\_usdhc\_auto\_command12\_error\_status\_flag).

**45.7.16 static uint32\_t USDHC\_GetAdmaErrorStatusFlags ( USDHC\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

ADMA error status flags mask(\_usdhc\_adma\_error\_status\_flag).

---

## Function Documentation

**45.7.17** `static uint32_t USDHC_GetPresentStatusFlags ( USDHC_Type * base )`  
`[inline], [static]`

This function gets the present USDHC's status except for an interrupt status and an error status.

## Parameters

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

## Returns

Present USDHC's status flags mask(*\_usdhc\_present\_status\_flag*).

#### 45.7.18 void USDHC\_GetCapability ( USDHC\_Type \* *base*, usdhc\_capability\_t \* *capability* )

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>capability</i>	Structure to save capability information.

#### 45.7.19 static void USDHC\_ForceClockOn ( USDHC\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable/disable</i>	flag.

#### 45.7.20 uint32\_t USDHC\_SetSdClock ( USDHC\_Type \* *base*, uint32\_t *srcClock\_Hz*, uint32\_t *busClock\_Hz* )

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>srcClock_Hz</i>	USDHC source clock frequency united in Hz.
<i>busClock_Hz</i>	SD bus clock frequency united in Hz.

## Returns

The nearest frequency of *busClock\_Hz* configured to SD bus.

---

## Function Documentation

### 45.7.21 `bool USDHC_SetCardActive ( USDHC_Type * base, uint32_t timeout )`

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>timeout</i>	Timeout to initialize card.

## Return values

<i>true</i>	Set card active successfully.
<i>false</i>	Set card active failed.

**45.7.22** `static void USDHC_AssertHardwareReset ( USDHC_Type * base, bool high ) [inline], [static]`

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>I</i>	or 0 level

**45.7.23** `static void USDHC_SetDataBusWidth ( USDHC_Type * base, usdhc_data_bus_width_t width ) [inline], [static]`

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>width</i>	Data transfer width.

**45.7.24** `static void USDHC_WriteData ( USDHC_Type * base, uint32_t data ) [inline], [static]`

This function is used to implement the data transfer by Data Port instead of DMA.

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>data</i>	The data about to be sent.

---

## Function Documentation

**45.7.25** `static uint32_t USDHC_ReadData ( USDHC_Type * base ) [inline],  
[static]`

This function is used to implement the data transfer by Data Port instead of DMA.



## Parameters

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

## Returns

The data has been read.

**45.7.26 void USDHC\_SendCommand ( USDHC\_Type \* *base*, usdhc\_command\_t \* *command* )**

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>command</i>	configuration

**45.7.27 static void USDHC\_EnableWakeupEvent ( USDHC\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	Wakeup events mask(_usdhc_wakeup_event).
<i>enable</i>	True to enable, false to disable.

**45.7.28 static void USDHC\_CardDetectByData3 ( USDHC\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable/disable</i>	flag

**45.7.29 static bool USDHC\_DetectCardInsert ( USDHC\_Type \* *base* ) [inline], [static]**

## Function Documentation

### Parameters

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

**45.7.30** `static void USDHC_EnableSdioControl ( USDHC_Type * base, uint32_t mask, bool enable ) [inline], [static]`

### Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	SDIO card control flags mask(_usdhc_sdio_control_flag).
<i>enable</i>	True to enable, false to disable.

**45.7.31** `static void USDHC_SetContinueRequest ( USDHC_Type * base ) [inline], [static]`

### Parameters

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

**45.7.32** `static void USDHC_RequestStopAtBlockGap ( USDHC_Type * base, bool enable ) [inline], [static]`

### Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	true to stop at block gap, false to normal transfer

**45.7.33** `void USDHC_SetMmcBootConfig ( USDHC_Type * base, const usdhc_boot_config_t * config )`

### Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
```

```

config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);

```

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>config</i>	The MMC boot configuration information.

**45.7.34 static void USDHC\_EnableMmcBoot ( USDHC\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	True to enable, false to disable.

**45.7.35 static void USDHC\_SetForceEvent ( USDHC\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The force events mask(_usdhc_force_event).

**45.7.36 static void USDHC\_SelectVoltage ( USDHC\_Type \* *base*, bool *en18v* )**  
**[inline], [static]**

## Parameters

<i>base</i>	USDHC peripheral base address.
<i>true</i>	1.8V, false 3.0V

**45.7.37 static bool USDHC\_RequestTuningForSDR50 ( USDHC\_Type \* *base* )**  
**[inline], [static]**

## Function Documentation

Parameters

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

**45.7.38 static bool USDHC\_RequestReTuning ( USDHC\_Type \* *base* ) [inline], [static]**

Parameters

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

**45.7.39 static void USDHC\_EnableAutoTuning ( USDHC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable/disable</i>	flag

**45.7.40 static void USDHC\_SetRetuningTimer ( USDHC\_Type \* *base*, uint32\_t *counter* ) [inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>timer</i>	counter value

**45.7.41 void USDHC\_EnableAutoTuningForCmdAndData ( USDHC\_Type \* *base* )**

Parameters

---

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

#### 45.7.42 void USDHC\_EnableManualTuning ( USDHC\_Type \* *base*, bool *enable* )

Parameters

<i>base</i>	USDHC peripheral base address.
<i>tuning</i>	enable flag

#### 45.7.43 status\_t USDHC\_AdjustDelayForManualTuning ( USDHC\_Type \* *base*, uint32\_t *delay* )

Parameters

<i>base</i>	USDHC peripheral base address.
<i>delay</i>	setting configuration

Return values

<i>kStatus_Fail</i>	config the delay setting fail
<i>kStatus_Success</i>	config the delay setting success

#### 45.7.44 void USDHC\_EnableStandardTuning ( USDHC\_Type \* *base*, uint32\_t *tuningStartTap*, uint32\_t *step*, bool *enable* )

Parameters

<i>base</i>	USDHC peripheral base address.
<i>tuning</i>	start tap
<i>tuning</i>	step

## Function Documentation

<i>enable/disable</i>	flag
-----------------------	------

**45.7.45** `static uint32_t USDHC_GetExecuteStdTuningStatus ( USDHC_Type * base ) [inline], [static]`

Parameters

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

**45.7.46** `static uint32_t USDHC_CheckStdTuningResult ( USDHC_Type * base ) [inline], [static]`

Parameters

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

**45.7.47** `static uint32_t USDHC_CheckTuningError ( USDHC_Type * base ) [inline], [static]`

Parameters

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

**45.7.48** `void USDHC_EnableDDRMMode ( USDHC_Type * base, bool enable, uint32_t nibblePos )`

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable/disable</i>	flag

<i>nibble</i>	position
---------------	----------

#### 45.7.49 **status\_t USDHC\_TransferBlocking ( USDHC\_Type \* *base*, usdhc\_adma\_config\_t \* *dmaConfig*, usdhc\_transfer\_t \* *transfer* )**

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

There is no need to call the API 'USDHC\_TransferCreateHandle' when calling this API.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>adma</i>	configuration
<i>transfer</i>	Transfer content.

Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_USDHC_- PrepareAdmaDescriptor- Failed</i>	Prepare ADMA descriptor failed.
<i>kStatus_USDHC_Send- CommandFailed</i>	Send command failed.
<i>kStatus_USDHC_- TransferDataFailed</i>	Transfer data failed.
<i>kStatus_Success</i>	Operate successfully.

#### 45.7.50 **void USDHC\_TransferCreateHandle ( USDHC\_Type \* *base*, usdhc\_handle\_t \* *handle*, const usdhc\_transfer\_callback\_t \* *callback*, void \* *userData* )**

## Function Documentation

### Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle pointer.
<i>callback</i>	Structure pointer to contain all callback functions.
<i>userData</i>	Callback function parameter.

### 45.7.51 **status\_t USDHC\_TransferNonBlocking ( USDHC\_Type \* *base*, usdhc\_handle\_t \* *handle*, usdhc\_adma\_config\_t \* *dmaConfig*, usdhc\_transfer\_t \* *transfer* )**

This function sends a command and data and returns immediately. It doesn't wait the transfer complete or encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

### Note

Call the API 'USDHC\_TransferCreateHandle' when calling this API.

### Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle.
<i>adma</i>	configuration.
<i>transfer</i>	Transfer content.

### Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_USDHC_Busy- Transferring</i>	Busy transferring.
<i>kStatus_USDHC_- PrepareAdmaDescriptor- Failed</i>	Prepare ADMA descriptor failed.



<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

**45.7.52 void USDHC\_TransferHandleIRQ ( USDHC\_Type \* *base*, usdhc\_handle\_t \* *handle* )**

This function deals with the IRQs on the given host controller.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle.



## Chapter 46

# WDOG: Watchdog Timer Driver

### 46.1 Overview

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

### 46.2 Typical use case

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

### Data Structures

- struct `wdog_work_mode_t`  
*Defines WDOG work mode. [More...](#)*
- struct `wdog_config_t`  
*Describes WDOG configuration structure. [More...](#)*
- struct `wdog_test_config_t`  
*Describes WDOG test mode configuration structure. [More...](#)*

### Enumerations

- enum `wdog_clock_source_t` {  
    `kWDOG_LpoClockSource` = 0U,  
    `kWDOG_AlternateClockSource` = 1U }  
*Describes WDOG clock source.*
- enum `wdog_clock_prescaler_t` {  
    `kWDOG_ClockPrescalerDivide1` = 0x0U,  
    `kWDOG_ClockPrescalerDivide2` = 0x1U,  
    `kWDOG_ClockPrescalerDivide3` = 0x2U,  
    `kWDOG_ClockPrescalerDivide4` = 0x3U,  
    `kWDOG_ClockPrescalerDivide5` = 0x4U,  
    `kWDOG_ClockPrescalerDivide6` = 0x5U,  
    `kWDOG_ClockPrescalerDivide7` = 0x6U,  
    `kWDOG_ClockPrescalerDivide8` = 0x7U }  
*Describes the selection of the clock prescaler.*
- enum `wdog_test_mode_t` {  
    `kWDOG_QuickTest` = 0U,  
    `kWDOG_ByteTest` = 1U }  
*Describes WDOG test mode.*
- enum `wdog_tested_byte_t` {  
    `kWDOG_TestByte0` = 0U,  
    `kWDOG_TestByte1` = 1U,  
    `kWDOG_TestByte2` = 2U,

## Typical use case

```
kWDOG_TestByte3 = 3U }
```

*Describes WDOG tested byte selection in byte test mode.*

- enum `_wdog_interrupt_enable_t` { `kWDOG_InterruptEnable` = `WDOG_STCTRLH_IRQRSTEN_MASK` }

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

- enum `_wdog_status_flags_t` {  
`kWDOG_RunningFlag` = `WDOG_STCTRLH_WDOGEN_MASK`,  
`kWDOG_TimeoutFlag` = `WDOG_STCTRLH_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 the WDOG configuration structure.*
- void `WDOG_Init` (`WDOG_Type` \*base, const `wdog_config_t` \*config)  
*Initializes the WDOG.*
- void `WDOG_Deinit` (`WDOG_Type` \*base)  
*Shuts down the WDOG.*
- void `WDOG_SetTestModeConfig` (`WDOG_Type` \*base, `wdog_test_config_t` \*config)  
*Configures the WDOG functional test.*

## WDOG Functional Operation

- static void `WDOG_Enable` (`WDOG_Type` \*base)  
*Enables the WDOG module.*
- static void `WDOG_Disable` (`WDOG_Type` \*base)  
*Disables the WDOG module.*
- static void `WDOG_EnableInterrupts` (`WDOG_Type` \*base, `uint32_t` mask)  
*Enables the WDOG interrupt.*
- static void `WDOG_DisableInterrupts` (`WDOG_Type` \*base, `uint32_t` mask)  
*Disables the WDOG interrupt.*

- uint32\_t [WDOG\\_GetStatusFlags](#) (WDOG\_Type \*base)  
*Gets the WDOG all status flags.*
- void [WDOG\\_ClearStatusFlags](#) (WDOG\_Type \*base, uint32\_t mask)  
*Clears the WDOG flag.*
- static void [WDOG\\_SetTimeoutValue](#) (WDOG\_Type \*base, uint32\_t timeoutCount)  
*Sets the WDOG timeout value.*
- static void [WDOG\\_SetWindowValue](#) (WDOG\_Type \*base, uint32\_t windowValue)  
*Sets the WDOG window value.*
- static void [WDOG\\_Unlock](#) (WDOG\_Type \*base)  
*Unlocks the WDOG register written.*
- void [WDOG\\_Refresh](#) (WDOG\_Type \*base)  
*Refreshes the WDOG timer.*
- static uint16\_t [WDOG\\_GetResetCount](#) (WDOG\_Type \*base)  
*Gets the WDOG reset count.*
- static void [WDOG\\_ClearResetCount](#) (WDOG\_Type \*base)  
*Clears the WDOG reset count.*

## 46.3 Data Structure Documentation

### 46.3.1 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.*

### 46.3.2 struct wdog\_config\_t

#### Data Fields

- bool [enableWdog](#)  
*Enables or disables WDOG.*
- [wdog\\_clock\\_source\\_t](#) clockSource  
*Clock source select.*
- [wdog\\_clock\\_prescaler\\_t](#) prescaler  
*Clock prescaler value.*
- [wdog\\_work\\_mode\\_t](#) workMode  
*Configures WDOG work mode in debug stop and wait mode.*
- bool [enableUpdate](#)  
*Update write-once register enable.*
- bool [enableInterrupt](#)  
*Enables or disables WDOG interrupt.*
- bool [enableWindowMode](#)  
*Enables or disables WDOG window mode.*
- uint32\_t [windowValue](#)  
*Window value.*

## Enumeration Type Documentation

- [uint32\\_t timeoutValue](#)  
*Timeout value.*

### 46.3.3 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.*

## 46.4 Macro Definition Documentation

### 46.4.1 #define FSL\_WDOG\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 46.5 Enumeration Type Documentation

### 46.5.1 enum wdog\_clock\_source\_t

Enumerator

*kWDOG\_LpoClockSource* WDOG clock sourced from LPO.

*kWDOG\_AlternateClockSource* WDOG clock sourced from alternate clock source.

### 46.5.2 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.

### 46.5.3 enum wdog\_test\_mode\_t

Enumerator

*kWDOG\_QuickTest* Selects quick test.

*kWDOG\_ByteTest* Selects byte test.

### 46.5.4 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.

### 46.5.5 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.

### 46.5.6 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.

## 46.6 Function Documentation

### 46.6.1 void WDOG\_GetDefaultConfig ( wdog\_config\_t \* config )

This function initializes the WDOG configuration structure to default values. The default values are as follows.

## Function Documentation

```
* wdogConfig->enableWdog = true;
* wdogConfig->clockSource = kWDOG_LpoClockSource;
* wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
* wdogConfig->workMode.enableWait = true;
* wdogConfig->workMode.enableStop = false;
* wdogConfig->workMode.enableDebug = false;
* wdogConfig->enableUpdate = true;
* wdogConfig->enableInterrupt = false;
* wdogConfig->enableWindowMode = false;
* wdogConfig->windowValue = 0;
* wdogConfig->timeoutValue = 0xFFFFU;
*
```

### Parameters

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

### See Also

[wdog\\_config\\_t](#)

### 46.6.2 void WDOG\_Init ( WDOG\_Type \* *base*, const wdog\_config\_t \* *config* )

This function initializes the WDOG. When called, the WDOG runs according to the configuration. To reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

This is an example.

```
* wdog_config_t config;
* WDOG_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* config.enableUpdate = true;
* WDOG_Init(wdog_base, &config);
*
```

### Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The configuration of WDOG

### 46.6.3 void WDOG\_Deinit ( WDOG\_Type \* *base* )

This function shuts down the WDOG. Ensure that the WDOG\_STCTRLH.ALLOWUPDATE is 1 which indicates that the register update is enabled.



#### 46.6.4 void WDOG\_SetTestModeConfig ( WDOG\_Type \* *base*, wdog\_test\_config\_t \* *config* )

This function is used to configure the WDOG functional test. When called, the WDOG goes into test mode and runs according to the configuration. Ensure that the WDOG\_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

This is an example.

```
* wdog_test_config_t test_config;
* test_config.testMode = kWDOG_QuickTest;
* test_config.timeoutValue = 0xfffffu;
* WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The functional test configuration of WDOG

#### 46.6.5 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
-------------	------------------------------

#### 46.6.6 static void WDOG\_Disable ( WDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_STCTRLH register to disable the WDOG. It is a write-once register. Ensure that the WCT window is still open and that register has not been written to in this WCT while the function is called.

Parameters

---

## Function Documentation

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

### 46.6.7 static void WDOG\_EnableInterrupts ( WDOG\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function writes a value into the WDOG\_STCTRLH register to enable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The interrupts to enable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"><li>• kWDOG_InterruptEnable</li></ul>

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

This function writes a value into the WDOG\_STCTRLH register to disable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The interrupts to disable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"><li>• kWDOG_InterruptEnable</li></ul>

### 46.6.9 uint32\_t WDOG\_GetStatusFlags ( WDOG\_Type \* *base* )

This function gets all status flags.

This is an example for getting the Running Flag.

```
* uint32_t status;  
* status = WDOG_GetStatusFlags (wdog_base) &  
* kWDOG_RunningFlag;
```

\*

#### Parameters

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

#### Returns

State of the status flag: asserted (true) or not-asserted (false).

#### See Also

##### [\\_wdog\\_status\\_flags\\_t](#)

- true: a related status flag has been set.
- false: a related status flag is not set.

### 46.6.10 void WDOG\_ClearStatusFlags ( WDOG\_Type \* *base*, uint32\_t *mask* )

This function clears the WDOG status flag.

This is an example for clearing the timeout (interrupt) flag.

```
* WDOG_ClearStatusFlags (wdog_base, kWDOG_TimeoutFlag);
*
```

#### Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

### 46.6.11 static void WDOG\_SetTimeoutValue ( WDOG\_Type \* *base*, uint32\_t *timeoutCount* ) [inline], [static]

This function sets the timeout value. It should be ensured that the time-out value for the WDOG is always greater than 2xWCT time + 20 bus clock cycles. This function writes a value into WDOG\_TOVALH and WDOG\_TOVALL registers which are write-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

## Function Documentation

### Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

### 46.6.12 static void WDOG\_SetWindowValue ( WDOG\_Type \* *base*, uint32\_t *windowValue* ) [*inline*], [*static*]

This function sets the WDOG window value. This function writes a value into WDOG\_WINH and WDOG\_WINL registers which are write-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

### Parameters

<i>base</i>	WDOG peripheral base address
<i>windowValue</i>	WDOG window value.

### 46.6.13 static void WDOG\_Unlock ( WDOG\_Type \* *base* ) [*inline*], [*static*]

This function unlocks the WDOG register written. Before starting the unlock sequence and following configuration, disable the global interrupts. Otherwise, an interrupt may invalidate the unlocking sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

### Parameters

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

### 46.6.14 void WDOG\_Refresh ( WDOG\_Type \* *base* )

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

### Parameters

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

**46.6.15** `static uint16_t WDOG_GetResetCount ( WDOG_Type * base ) [inline],  
[static]`

This function gets the WDOG reset count value.

## Function Documentation

Parameters

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

Returns

WDOG reset count value.

**46.6.16** `static void WDOG_ClearResetCount ( WDOG_Type * base ) [inline],  
[static]`

This function clears the WDOG reset count value.

Parameters

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

# Chapter 47

## XBARA: Inter-Peripheral Crossbar Switch

### 47.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARA) block of MCUXpresso SDK devices.

The XBARA peripheral driver configures the XBARA (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARA module.

XBARA driver has two parts:

- Signal connection interconnects input and output signals.
- Active edge feature - Some of the outputs provide an active edge detection. If an active edge occurs, an interrupt or a DMA request can be called. APIs handle user callbacks for the interrupts. The driver also includes API for clearing and reading status bits.

### 47.2 Function

#### 47.2.1 XBARA Initialization

To initialize the XBARA driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARA module is initialized by calling the [XBARA\\_Init\(\)](#) function.

#### 47.2.2 Call diagram

1. Call the "XBARA\_Init()" function to initialize the XBARA module.
2. Optionally, call the "XBARA\_SetSignalsConnection()" function to Set connection between the selected XBARA\_IN[\*] input and the XBARA\_OUT[\*] output signal. It connects the XBARA input to the selected XBARA output. A configuration structure of the "xbara\_input\_signal\_t" type and "xbara\_output\_signal\_t" type is required.
3. Call the "XBARA\_SetOutputSignalConfig" function to set the active edge features, such interrupts or DMA requests. A configuration structure of the "xbara\_control\_config\_t" type is required to point to structure that keeps configuration of control register.
4. Finally, the XBARA works properly.

### 47.3 Typical use case

#### Data Structures

- struct [xbara\\_control\\_config\\_t](#)  
*Defines the configuration structure of the XBARA control register. [More...](#)*

## Data Structure Documentation

### Macros

- #define `FSL_XBARA_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 3))  
*Version 2.0.3.*

### Enumerations

- enum `xbara_active_edge_t` {  
    `kXBARA_EdgeNone` = 0U,  
    `kXBARA_EdgeRising` = 1U,  
    `kXBARA_EdgeFalling` = 2U,  
    `kXBARA_EdgeRisingAndFalling` = 3U }  
*XBARA active edge for detection.*
- enum `xbara_request_t` {  
    `kXBARA_RequestDisable` = 0U,  
    `kXBARA_RequestDMAEnable` = 1U,  
    `kXBARA_RequestInterruptEnable` = 2U }  
*Defines the XBARA DMA and interrupt configurations.*
- enum `xbara_status_flag_t` {  
    `kXBARA_EdgeDetectionOut0`,  
    `kXBARA_EdgeDetectionOut1`,  
    `kXBARA_EdgeDetectionOut2`,  
    `kXBARA_EdgeDetectionOut3` }  
*XBARA status flags.*

### XBARA functional Operation.

- void `XBARA_Init` (XBARA\_Type \*base)  
*Initializes the XBARA module.*
- void `XBARA_Deinit` (XBARA\_Type \*base)  
*Shuts down the XBARA module.*
- void `XBARA_SetSignalsConnection` (XBARA\_Type \*base, xbar\_input\_signal\_t input, xbar\_output\_signal\_t output)  
*Sets a connection between the selected XBARA\_IN[\*] input and the XBARA\_OUT[\*] output signal.*
- uint32\_t `XBARA_GetStatusFlags` (XBARA\_Type \*base)  
*Gets the active edge detection status.*
- void `XBARA_ClearStatusFlags` (XBARA\_Type \*base, uint32\_t mask)  
*Clears the edge detection status flags of relative mask.*
- void `XBARA_SetOutputSignalConfig` (XBARA\_Type \*base, xbar\_output\_signal\_t output, const `xbara_control_config_t` \*controlConfig)  
*Configures the XBARA control register.*

## 47.4 Data Structure Documentation

### 47.4.1 struct `xbara_control_config_t`

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.



## Data Fields

- [xbara\\_active\\_edge\\_t](#) `activeEdge`  
*Active edge to be detected.*
- [xbara\\_request\\_t](#) `requestType`  
*Selects DMA/Interrupt request.*

### 47.4.1.0.0.61 Field Documentation

47.4.1.0.0.61.1 [xbara\\_active\\_edge\\_t](#) `xbara_control_config_t::activeEdge`

47.4.1.0.0.61.2 [xbara\\_request\\_t](#) `xbara_control_config_t::requestType`

## 47.5 Macro Definition Documentation

47.5.1 `#define FSL_XBARA_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`

## 47.6 Enumeration Type Documentation

### 47.6.1 `enum xbara_active_edge_t`

Enumerator

- kXBARA\_EdgeNone*** Edge detection status bit never asserts.
- kXBARA\_EdgeRising*** Edge detection status bit asserts on rising edges.
- kXBARA\_EdgeFalling*** Edge detection status bit asserts on falling edges.
- kXBARA\_EdgeRisingAndFalling*** Edge detection status bit asserts on rising and falling edges.

### 47.6.2 `enum xbara_request_t`

Enumerator

- kXBARA\_RequestDisable*** Interrupt and DMA are disabled.
- kXBARA\_RequestDMAEnable*** DMA enabled, interrupt disabled.
- kXBARA\_RequestInterruptEnable*** Interrupt enabled, DMA disabled.

### 47.6.3 `enum xbara_status_flag_t`

This provides constants for the XBARA status flags for use in the XBARA functions.

Enumerator

- kXBARA\_EdgeDetectionOut0*** XBAR\_OUT0 active edge interrupt flag, sets when active edge detected.

## Function Documentation

***kXBARA\_EdgeDetectionOut1*** XBAR\_OUT1 active edge interrupt flag, sets when active edge detected.

***kXBARA\_EdgeDetectionOut2*** XBAR\_OUT2 active edge interrupt flag, sets when active edge detected.

***kXBARA\_EdgeDetectionOut3*** XBAR\_OUT3 active edge interrupt flag, sets when active edge detected.

## 47.7 Function Documentation

### 47.7.1 void XBARA\_Init ( XBARA\_Type \* *base* )

This function un-gates the XBARA clock.

Parameters

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

### 47.7.2 void XBARA\_Deinit ( XBARA\_Type \* *base* )

This function disables XBARA clock.

Parameters

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

### 47.7.3 void XBARA\_SetSignalsConnection ( XBARA\_Type \* *base*, xbar\_input\_signal\_t *input*, xbar\_output\_signal\_t *output* )

This function connects the XBARA input to the selected XBARA output. If more than one XBARA module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBARA_SetSignalsConnection(XBARA, kXBARA_InputPIT_TRG0, kXBARA_OutputDMAMUX18);
```

Parameters

---

<i>base</i>	XBARA peripheral address.
<i>input</i>	XBARA input signal.
<i>output</i>	XBARA output signal.

#### 47.7.4 uint32\_t XBARA\_GetStatusFlags ( XBARA\_Type \* *base* )

This function gets the active edge detect status of all XBARA\_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBARA\_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

<i>base</i>	XBARA peripheral address.
-------------	---------------------------

Returns

the mask of these status flag bits.

#### 47.7.5 void XBARA\_ClearStatusFlags ( XBARA\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	XBARA peripheral address.
<i>mask</i>	the status flags to clear.

#### 47.7.6 void XBARA\_SetOutputSignalConfig ( XBARA\_Type \* *base*, xbar\_output\_signal\_t *output*, const xbara\_control\_config\_t \* *controlConfig* )

This function configures an XBARA control register. The active edge detection and the DMA/IRQ function on the corresponding XBARA output can be set.

Example:

```
xbara_control_config_t userConfig;
userConfig.activeEdge = kXBARA_EdgeRising;
userConfig.requestType = kXBARA_RequestInterruptEnalbe;
XBARA_SetOutputSignalConfig(XBARA, kXBARA_OutputDMAMUX18, &userConfig);
```

## Function Documentation

### Parameters

<i>base</i>	XBARA peripheral address.
<i>output</i>	XBARA output number.
<i>controlConfig</i>	Pointer to structure that keeps configuration of control register.

# Chapter 48

## XBARB: Inter-Peripheral Crossbar Switch

### 48.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARB) block of MCUXpresso SDK devices.

The XBARB peripheral driver configures the XBARB (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARB module.

XBARB driver has two parts:

- Signal connection interconnects input and output signals.

### 48.2 Function groups

#### 48.2.1 XBARB Initialization

To initialize the XBARB driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARB module is initialized by calling the [XBARB\\_Init\(\)](#) function.

#### 48.2.2 Call diagram

1. Call the "XBARB\_Init()" function to initialize the XBARB module.
2. Optionally, call the "XBARB\_SetSignalsConnection()" function to Set connection between the selected XBARB\_IN[\*] input and the XBARB\_OUT[\*] output signal. It connects the XBARB input to the selected XBARB output. A configuration structure of the "xbarb\_input\_signal\_t" type and "xbarb\_output\_signal\_t" type is required.
3. Finally, the XBARB works properly.

### 48.3 Typical use case

#### Macros

- #define [FSL\\_XBARB\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 1))  
*Version 2.0.1.*

#### XBARB functional Operation.

- void [XBARB\\_Init](#) (XBARB\_Type \*base)  
*Initializes the XBARB module.*
- void [XBARB\\_Deinit](#) (XBARB\_Type \*base)

## Function Documentation

*Shuts down the XBARB module.*

- void [XBARB\\_SetSignalsConnection](#) (XBARB\_Type \*base, xbar\_input\_signal\_t input, xbar\_output\_signal\_t output)

*Configures a connection between the selected XBARB\_IN[\*] input and the XBARB\_OUT[\*] output signal.*

## 48.4 Macro Definition Documentation

### 48.4.1 #define FSL\_XBARB\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 48.5 Function Documentation

### 48.5.1 void XBARB\_Init ( XBARB\_Type \* *base* )

This function un-gates the XBARB clock.

Parameters

<i>base</i>	XBARB peripheral address.
-------------	---------------------------

### 48.5.2 void XBARB\_Deinit ( XBARB\_Type \* *base* )

This function disables XBARB clock.

Parameters

<i>base</i>	XBARB peripheral address.
-------------	---------------------------

### 48.5.3 void XBARB\_SetSignalsConnection ( XBARB\_Type \* *base*, xbar\_input\_signal\_t *input*, xbar\_output\_signal\_t *output* )

This function configures which XBARB input is connected to the selected XBARB output. If more than one XBARB module is available, only the inputs and outputs from the same module can be connected.

Parameters

<i>base</i>	XBARB peripheral address.
<i>input</i>	XBARB input signal.

<i>output</i>	XBARB output signal.
---------------	----------------------







## **Chapter 49**

### **Clock Driver**

The MCUXpresso SDK provides APIs for MCUXpresso SDK devices' clock operation.



# Chapter 50

## DMA Manager

### 50.1 Overview

DMA Manager provides a series of functions to manage the DMAMUX instances and channels.

### 50.2 Function groups

#### 50.2.1 DMAMGR Initialization and De-initialization

This function group initializes and deinitializes the DMA Manager.

#### 50.2.2 DMAMGR Operation

This function group requests/releases the DMAMUX channel and configures the channel request source.

### 50.3 Typical use case

#### 50.3.1 DMAMGR static channel allocation

```
uint8_t channel;
dmamanager_handle_t dmamanager_handle;

/* Initialize DMAMGR */
DMAMGR_Init(&dmamanager_handle, EXAMPLE_DMA_BASEADDR, DMA_CHANNEL_NUMBER, startChannel);
/* Request a DMAMUX channel by static allocate mechanism */
channel = kDMAMGR_STATIC_ALLOCATE;
DMAMGR_RequestChannel(&dmamanager_handle, kDmaRequestMux0AlwaysOn63, channel, &handle)
    ;
```

#### 50.3.2 DMAMGR dynamic channel allocation

```
uint8_t channel;
dmamanager_handle_t dmamanager_handle;

/* Initialize DMAMGR */
DMAMGR_Init(&dmamanager_handle, EXAMPLE_DMA_BASEADDR, DMA_CHANNEL_NUMBER, startChannel);
/* Request a DMAMUX channel by Dynamic allocate mechanism */
channel = DMAMGR_DYNAMIC_ALLOCATE;
DMAMGR_RequestChannel(&dmamanager_handle, kDmaRequestMux0AlwaysOn63, channel, &handle)
    ;
```

### Data Structures

- struct `dmamanager_handle_t`  
*dmamanager handle typedef. [More...](#)*

## Data Structure Documentation

### Macros

- #define `DMAMGR_DYNAMIC_ALLOCATE` 0xFFU  
*Dynamic channel allocation 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) }  
*DMA manager status.*

### DMAMGR Initialization and De-initialization

- void `DMAMGR_Init` (`dmamanager_handle_t` \*dmamanager\_handle, `DMA_Type` \*dma\_base, `uint32_t` channelNum, `uint32_t` startChannel)  
*Initializes the DMA manager.*
- void `DMAMGR_Deinit` (`dmamanager_handle_t` \*dmamanager\_handle)  
*Deinitializes the DMA manager.*

### DMAMGR Operation

- `status_t` `DMAMGR_RequestChannel` (`dmamanager_handle_t` \*dmamanager\_handle, `uint32_t` requestSource, `uint32_t` channel, void \*handle)  
*Requests a DMA channel.*
- `status_t` `DMAMGR_ReleaseChannel` (`dmamanager_handle_t` \*dmamanager\_handle, void \*handle)  
*Releases a DMA channel.*
- `bool` `DMAMGR_IsChannelOccupied` (`dmamanager_handle_t` \*dmamanager\_handle, `uint32_t` channel)  
*Get a DMA channel status.*

## 50.4 Data Structure Documentation

### 50.4.1 struct dmamanager\_handle\_t

Note

The contents of this structure are private and subject to change.

This dma manager handle structure is used to store the parameters transferred by users. And users shall not free the memory before calling `DMAMGR_Deinit`, also shall not modify the contents of the memory.

### Data Fields

- void \* `dma_base`  
*Peripheral DMA instance.*
- `uint32_t` `channelNum`

- Channel numbers for the DMA instance which need to be managed by dma manager.*

  - uint32\_t [startChannel](#)  
*The start channel that can be managed by dma manager,users need to transfer it with a certain number or NULL.*
  - bool [s\\_DMAMGR\\_Channels](#) [64]  
*The s\_DMAMGR\_Channels is used to store dma manager state.*
  - uint32\_t [DmamuxInstanceStart](#)  
*The DmamuxInstance is used to calculate the DMAMUX Instance according to the DMA Instance.*
  - uint32\_t [multiple](#)  
*The multiple is used to calculate the multiple between DMAMUX count and DMA count.*

### 50.4.1.0.0.62 Field Documentation

50.4.1.0.0.62.1 void\* [dmamanager\\_handle\\_t::dma\\_base](#)

50.4.1.0.0.62.2 uint32\_t [dmamanager\\_handle\\_t::channelNum](#)

50.4.1.0.0.62.3 uint32\_t [dmamanager\\_handle\\_t::startChannel](#)

50.4.1.0.0.62.4 bool [dmamanager\\_handle\\_t::s\\_DMAMGR\\_Channels\[64\]](#)

50.4.1.0.0.62.5 uint32\_t [dmamanager\\_handle\\_t::DmamuxInstanceStart](#)

50.4.1.0.0.62.6 uint32\_t [dmamanager\\_handle\\_t::multiple](#)

## 50.5 Macro Definition Documentation

50.5.1 #define [DMAMGR\\_DYNAMIC\\_ALLOCATE 0xFFU](#)

## 50.6 Enumeration Type Documentation

50.6.1 enum [\\_dma\\_manager\\_status](#)

Enumerator

*kStatus\_DMAMGR\_ChannelOccupied* Channel has been occupied.

*kStatus\_DMAMGR\_ChannelNotUsed* Channel has not been used.

*kStatus\_DMAMGR\_NoFreeChannel* All channels have been occupied.

## 50.7 Function Documentation

50.7.1 void [DMAMGR\\_Init](#) ( [dmamanager\\_handle\\_t](#) \* *dmamanager\_handle*,  
[DMA\\_Type](#) \* *dma\_base*, [uint32\\_t](#) *channelNum*, [uint32\\_t](#) *startChannel* )

This function initializes the DMA manager, ungates the DMAMUX clocks, and initializes the eDMA or DMA peripherals.

## Function Documentation

### Parameters

<i>dmamanager_handle</i>	DMA manager handle pointer, this structure is maintained by dma manager internal, users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
<i>dma_base</i>	Peripheral DMA instance base pointer.
<i>dmamux_base</i>	Peripheral DMAMUX instance base pointer.
<i>channelNum</i>	Channel numbers for the DMA instance which need to be managed by dma manager.
<i>startChannel</i>	The start channel that can be managed by dma manager.

### 50.7.2 void DMAMGR\_Deinit ( dmamanager\_handle\_t \* dmamanager\_handle )

This function deinitializes the DMA manager, disables the DMAMUX channels, gates the DMAMUX clocks, and deinitializes the eDMA or DMA peripherals.

### Parameters

<i>dmamanager_handle</i>	DMA manager handle pointer, this structure is maintained by dma manager internal, users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
--------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### 50.7.3 status\_t DMAMGR\_RequestChannel ( dmamanager\_handle\_t \* dmamanager\_handle, uint32\_t requestSource, uint32\_t channel, void \* handle )

This function requests a DMA channel which is not occupied. The two channels to allocate the mechanism are dynamic and static channels. For the dynamic allocation mechanism (channe = DMAMGR\_DYNAMIC\_ALLOCATE), DMAMGR allocates a DMA channel according to the given request source and start channel and then configures it. For static allocation mechanism, DMAMGR configures the given channel according to the given request source and channel number.

Parameters

<i>dmamanager_-handle</i>	DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
<i>requestSource</i>	DMA channel request source number. See the soc.h, see the enum dma_request_source_t
<i>channel</i>	The channel number users want to occupy. If using the dynamic channel allocate mechanism, set the channel equal to DMAMGR_DYNAMIC_ALLOCATE.
<i>handle</i>	DMA or eDMA handle pointer.

Return values

<i>kStatus_Success</i>	In a dynamic/static channel allocation mechanism, allocate the DMAMUX channel successfully.
<i>kStatus_DMAMGR_NoFreeChannel</i>	In a dynamic channel allocation mechanism, all DMAMUX channels are occupied.
<i>kStatus_DMAMGR_ChannelOccupied</i>	In a static channel allocation mechanism, the given channel is occupied.

**50.7.4 status\_t DMAMGR\_ReleaseChannel ( dmamanager\_handle\_t \* dmamanager\_handle, void \* handle )**

This function releases an occupied DMA channel.

Parameters

<i>dmamanager_-handle</i>	DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
<i>handle</i>	DMA or eDMA handle pointer.

Return values

---

## Function Documentation

<i>kStatus_Success</i>	Releases the given channel successfully.
<i>kStatus_DMAMGR_ChannelNotUsed</i>	The given channel to be released had not been used before.

### 50.7.5 **bool DMAMGR\_IsChannelOccupied ( dmamanager\_handle\_t \* dmamanager\_handle, uint32\_t channel )**

This function get a DMA channel status. Return 0 indicates the channel has not been used, return 1 indicates the channel has been occupied.

Parameters

<i>dmamanager_handle</i>	DMA manager handle pointer, this structure is maintained by dma manager internal, users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
<i>channel</i>	The channel number that users want get its status.



# Chapter 51

## Debug Console

### 51.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. It consists of log, str, io. Log layer is used to handle the formatted log, push log to buffer or flush log to IO. STR layer is used to format the printf and scanf log. IO layer is a wrapper of various uart peripheral.

### 51.2 Function groups

#### 51.2.1 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 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.

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);
```

## Function groups

### 51.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype "`%[flags][width][.precision][length]specifier`", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

<b>.precision</b>	<b>Description</b>
.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.

<b>length</b>	<b>Description</b>
Do not support	

<b>specifier</b>	<b>Description</b>
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. In other words, it is not stored in the corresponding argument.

width	Description
	This specifies the maximum number of characters to be read in the current reading operation.

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *

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 MCUXpresso SDK 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 */
```

## 51.3 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 equivalent 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\r\nTime: %u ticks %2.5f milliseconds\r\n\r\n", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");  
SCANF("%d", &i);  
PRINTF("\r\nYou have entered %d.\r\n", i, i);  
PRINTF("Enter a hexadecimal number: ");  
SCANF("%x", &i);  
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

### Print out failure messages using MCUXpresso SDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)  
{  
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file ,  
        line, func);  
    for (;;)   
    {}  
}
```

### Note:

To use 'printf' and 'scanf' for GNUC Base, add file 'fsl\_sbrk.c' in path: ..\{package}\devices\{subset}\utilities\fsl-  
\_sbrk.c to your project.

### Modules

- [SWO](#)  
/\*!
- [Semihosting](#)

### Data Structures

- struct [io\\_state\\_t](#)  
*State structure storing io. [More...](#)*

## Macros

- #define **SDK\_DEBUGCONSOLE** 1U  
*Definition to select sdk or toolchain printf, scanf.*
- #define **SDK\_DEBUGCONSOLE\_UART**  
*Definition to select redirect toolchain printf, scanf to uart or not.*

## Typedefs

- typedef void(\* **notify** )(size\_t \*size, bool rx, bool tx)  
*define a notify callback for IO*
- typedef void(\* **printfCb** )(char \*buf, int32\_t \*indicator, char val, int len)  
*A function pointer which is used when format printf log.*

## Enumerations

- enum **\_swo\_protocol** {  
    **kSWO\_ProtocolManchester** = 1U,  
    **kSWO\_ProtocolNrz** = 2U }

## Functions

- void **IO\_Init** (io\_state\_t \*io, uint32\_t baudRate, uint32\_t clkSrcFreq, uint8\_t \*ringBuffer)  
*io init function.*
- status\_t **IO\_Deinit** (void)  
*Deinit IO.*
- status\_t **IO\_Transfer** (uint8\_t \*ch, size\_t size, bool tx)  
*io transfer function.*
- status\_t **IO\_WaitIdle** (void)  
*io wait idle.*
- status\_t **SWO\_Init** (uint32\_t port, uint32\_t baudRate, uint32\_t clkSrcFreq)  
*io init function.*
- void **SWO\_Deinit** (uint32\_t port)  
*Deinit IO.*
- status\_t **SWO\_SendBlocking** (uint32\_t port, uint8\_t \*ch, size\_t size)  
*io transfer function.*
- status\_t **LOG\_Init** (uint32\_t baseAddr, uint8\_t device, uint32\_t baudRate, uint32\_t clkSrcFreq)  
*Initializes.*
- void **LOG\_Deinit** (void)  
*De-Initializes.*
- int **LOG\_Push** (uint8\_t \*buf, size\_t size)  
*log push interface*
- int **LOG\_ReadLine** (uint8\_t \*buf, size\_t size)  
*log read one line function*
- int **LOG\_ReadCharacter** (uint8\_t \*ch)  
*log read one character function*
- status\_t **LOG\_WaitIdle** (void)  
*wait log and io idle*
- int **LOG\_Pop** (uint8\_t \*buf, size\_t size)  
*log pop function*
- int **StrFormatPrintf** (const char \*fmt, va\_list ap, char \*buf, printfCb cb)

## Typedef Documentation

*This function outputs its parameters according to a formatted string.*

- int **StrFormatScanf** (const char \*line\_ptr, char \*format, va\_list args\_ptr)  
*Converts an input line of ASCII characters based upon a provided string format.*

## Initialization

- status\_t **DbgConsole\_Init** (uint32\_t baseAddr, uint32\_t baudRate, uint8\_t device, uint32\_t clkSrcFreq)  
*Initializes the peripheral used for debug messages.*
- status\_t **DbgConsole\_Deinit** (void)  
*De-initializes the peripheral used for debug messages.*
- int **DbgConsole\_Printf** (const char \*fmt\_s,...)  
*Writes formatted output to the standard output stream.*
- int **DbgConsole\_Putchar** (int ch)  
*Writes a character to stdout.*
- int **DbgConsole\_Scanf** (char \*fmt\_ptr,...)  
*Reads formatted data from the standard input stream.*
- int **DbgConsole\_Getchar** (void)  
*Reads a character from standard input.*
- status\_t **DbgConsole\_Flush** (void)  
*Debug console flush log.*

## 51.4 Data Structure Documentation

### 51.4.1 struct io\_state\_t

#### Data Fields

- void \* **ioBase**  
*Base of the IP register.*
- uint8\_t **ioType**  
*device type*

#### 51.4.1.0.0.63 Field Documentation

##### 51.4.1.0.0.63.1 void\* io\_state\_t::ioBase

## 51.5 Macro Definition Documentation

### 51.5.1 #define SDK\_DEBUGCONSOLE 1U

### 51.5.2 #define SDK\_DEBUGCONSOLE\_UART

## 51.6 Typedef Documentation

### 51.6.1 typedef void(\* notify)(size\_t \*size, bool rx, bool tx)



Parameters

<i>size,transfer</i>	data size.
<i>rx,indicate</i>	a rx transfer is success.
<i>tx,indicate</i>	a tx transfer is success.

## 51.7 Enumeration Type Documentation

### 51.7.1 enum \_swo\_protocol

Enumerator

*kSWO\_ProtocolManchester* SWO manchester protocol.

*kSWO\_ProtocolNrz* SWO UART/NRZ protocol.

## 51.8 Function Documentation

### 51.8.1 status\_t DbgConsole\_Init ( uint32\_t baseAddr, uint32\_t baudRate, uint8\_t device, uint32\_t clkSrcFreq )

Call this function to enable debug log messages to be output via the specified peripheral, frequency of peripheral source clock, and base address at the specified baud rate. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

<i>baseAddr</i>	Indicates the address of the peripheral used to send debug messages.
<i>baudRate</i>	The desired baud rate in bits per second.
<i>device</i>	Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"> <li>• DEBUG_CONSOLE_DEVICE_TYPE_UART,</li> <li>• DEBUG_CONSOLE_DEVICE_TYPE_LPUART,</li> <li>• DEBUG_CONSOLE_DEVICE_TYPE_LPSCI,</li> <li>• DEBUG_CONSOLE_DEVICE_TYPE_USBCDC.</li> </ul>

## Function Documentation

<i>clkSrcFreq</i>	Frequency of peripheral source clock.
-------------------	---------------------------------------

Returns

Indicates whether initialization was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully
<i>kStatus_Fail</i>	Execution failure
<i>kStatus_InvalidArgument</i>	Invalid argument existed

### 51.8.2 **status\_t DbgConsole\_Deinit ( void )**

Call this function to disable debug log messages to be output via the specified peripheral base address and at the specified baud rate.

Returns

Indicates whether de-initialization was successful or not.

### 51.8.3 **int DbgConsole\_Printf ( const char \* *fmt\_s*, ... )**

Call this function to write a formatted output to the standard output stream.

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 51.8.4 **int DbgConsole\_Putchar ( int *ch* )**

Call this function to write a character to stdout.

## Parameters

<i>ch</i>	Character to be written.
-----------	--------------------------

## Returns

Returns the character written.

**51.8.5 int DbgConsole\_Scanf ( char \* *fmt\_ptr*, ... )**

Call this function to read formatted data from the standard input stream.

## Parameters

<i>fmt_ptr</i>	Format control string.
----------------	------------------------

## Returns

Returns the number of fields successfully converted and assigned.

**51.8.6 int DbgConsole\_Getchar ( void )**

Call this function to read a character from standard input.

## Returns

Returns the character read.

**51.8.7 status\_t DbgConsole\_Flush ( void )**

Call this function to wait the buffer empty and io idle before. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

## Returns

Indicates whether wait idle was successful or not.

**51.8.8 void IO\_Init ( io\_state\_t \* *io*, uint32\_t *baudRate*, uint32\_t *clkSrcFreq*, uint8\_t \* *ringBuffer* )**

Call this function to init IO.

## Function Documentation

Parameters

<i>io</i>	configuration pointer
<i>baudRate</i>	baud rate
<i>clkSrcFreq</i>	clock freq
<i>ringbuffer</i>	used to receive character

### 51.8.9 status\_t IO\_Deinit ( void )

Call this function to Deinit IO.

Returns

deinit status

### 51.8.10 status\_t IO\_Transfer ( uint8\_t \* *ch*, size\_t *size*, bool *tx* )

Call this function to transfer log. Print log:

```
* IO_Transfer(ch, size, true);  
*
```

Scanf log:

```
* IO_Transfer(ch, size, false);  
*
```

Parameters

<i>ch</i>	transfer buffer pointer
<i>size</i>	transfer size
<i>tx</i>	indicate the transfer is TX or RX

### 51.8.11 status\_t IO\_WaitIdle ( void )

Call this function to wait the io idle

Returns

Indicates whether wait idle was successful or not.

**51.8.12** `status_t SWO_Init ( uint32_t port, uint32_t baudRate, uint32_t clkSrcFreq )`

Call this function to init SWO.

## Function Documentation

Parameters

<i>port</i>	port used to transfer data
<i>baudRate</i>	SWO clock
<i>clkSrcFreq</i>	core clock frequency

### 51.8.13 void SWO\_Deinit ( uint32\_t port )

Call this function to Deinit SWO.

Parameters

<i>port</i>	port to deinit.
-------------	-----------------

Returns

deinit status

### 51.8.14 status\_t SWO\_SendBlocking ( uint32\_t port, uint8\_t \* ch, size\_t size )

Call this function to print log.

Parameters

<i>port</i>	port used to transfer data
<i>ch</i>	transfer buffer pointer
<i>size</i>	transfer size

### 51.8.15 status\_t LOG\_Init ( uint32\_t baseAddr, uint8\_t device, uint32\_t baudRate, uint32\_t clkSrcFreq )

Call this function to init the buffer

Parameters

<i>base-Addr,device</i>	base address
<i>device,device</i>	type
<i>baud-Rate,device</i>	communicate baudrate
<i>clkSrc-Freq,device</i>	source clock freq

Returns

Indicates whether initialization was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully
<i>kStatus_Fail</i>	Execution failure

### 51.8.16 void LOG\_Deinit ( void )

Call this function to deinit the buffer

Returns

Indicates whether Deinit was successful or not.

### 51.8.17 int LOG\_Push ( uint8\_t \* buf, size\_t size )

Call this function to print log

Parameters

<i>fmt,buffer</i>	pointer
<i>size,available</i>	size

Returns

indicate the push size

## Function Documentation

Return values

	indicate buffer is full or transfer fail.
<i>size</i>	return the push log size.

### 51.8.18 int LOG\_ReadLine ( uint8\_t \* *buf*, size\_t *size* )

Call this function to print log

Parameters

<i>fmt,buffer</i>	pointer
<i>size,available</i>	size the number of the recieved character

### 51.8.19 int LOG\_ReadCharacter ( uint8\_t \* *ch* )

Call this function to GETCHAR

Parameters

<i>ch</i>	receive address the number of the recieved character
-----------	------------------------------------------------------

### 51.8.20 status\_t LOG\_WaitIdle ( void )

Call this function to wait log buffer empty and io idle before enter low power mode.

Returns

Indicates whether wait idle was successful or not.

### 51.8.21 int LOG\_Pop ( uint8\_t \* *buf*, size\_t *size* )

Call this function to pop log from buffer.



## Parameters

<i>buf</i>	buffer address to pop
<i>size</i>	log size to pop

## Returns

pop log size.

### 51.8.22 int StrFormatPrintf ( const char \* *fmt*, va\_list *ap*, char \* *buf*, printfCb *cb* )

## Note

I/O is performed by calling given function pointer using following (\*func\_ptr)(c);

## Parameters

in	<i>fmt_ptr</i>	Format string for printf.
in	<i>args_ptr</i>	Arguments to printf.
in	<i>buf</i>	pointer to the buffer
	<i>cb</i>	print callbck function pointer

## Returns

Number of characters to be print

### 51.8.23 int StrFormatScanf ( const char \* *line\_ptr*, char \* *format*, va\_list *args\_ptr* )

## Parameters

in	<i>line_ptr</i>	The input line of ASCII data.
in	<i>format</i>	Format first points to the format string.
in	<i>args_ptr</i>	The list of parameters.

## Returns

Number of input items converted and assigned.

---

## Function Documentation

Return values

<i>IO_EOF</i>	When line_ptr is empty string "".
---------------	-----------------------------------

## 51.9 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

### 51.9.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the `SDK_DEBUGCONSOLE` is disabled.

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

#### Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

#### Step 3: Starting semihosting

1. Choose "Semihosting\_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
  2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
  3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting.
1. Make sure the `SDK_DEBUGCONSOLE_UART` is not defined, remove the default definition in `fsl_debug_console.h`.
  1. Start the project by choosing Project>Download and Debug.
  2. Choose View>Terminal I/O to display the output from the I/O operations.

### 51.9.2 Guide Semihosting for Keil $\mu$ Vision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

## Semihosting

### Step 1: Setting up the environment

1. Make sure the SDK\_DEBUGCONSOLE\_UART is not defined, remove the default definition in fsl\_debug\_console.h..
2. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
3. Open Project>Options for target or using Alt+F7 or click.
4. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
5. Select “Debug” tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

### Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

### Step 4: 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.

## 51.9.3 Guide Semihosting for MCUXpresso IDE

### Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK\_DEBUGCONSOLE=0, if set SDK\_DEBUGCONSOLE=1, the log will be redirect to the UART.

### Step 2: Building the project

1. Compile and link the project.

### Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

## 51.9.4 Guide Semihosting for ARMGCC

### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__stack_size__=0x2000")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__stack_size__=0x2000")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__heap_size__=0x2000")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__heap_size__=0x2000")
```

### Step 2: Building the project

1. Change "CMakeLists.txt":
 

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"

**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

#### Replace paragraph

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fno-common")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-ffunction-sections")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fdata-sections")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-ffreestanding")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fno-builtin")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-mthumb")
```

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-mapcs")
```

## Semihosting

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} --gc-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -static")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} muldefs")
```

### To

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
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 as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\trk64f120m\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 now shows the printf() output.

## 51.10 SWO

/\*!

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDE support SWO also, such IAR and KEIL, both input and output are supported, reference below for detail.

### 51.10.1 Guide SWO for SDK

**NOTE:** After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

#### Step 1: Setting up the environment

1. Define `DEBUG_CONSOLE_IO_SWO` in your project settings.
2. Prepare code, the port and baudrate can be decided by application, `clkSrcFreq` should be mcu core clock frequency:

```
DbgConsole_Init(port, baudrate, DEBUG_CONSOLE_DEVICE_TYPE_SWO, clkSrcFreq);
```

3. Use `PRINTF` or `printf` to print some thing in application.

#### Step 2: Building the project

#### Step 3: Download and run project

### 51.10.1.1 Guide SWO for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
1. Make sure the `SDK_DEBUGCONSOLE_UART` is not defined, remove the default definition in `fsl_debug_console.h..`
1. Start the project by choosing Project>Download and Debug.

#### Step 2: Building the project

#### Step 3: Starting swo

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.

## SWO

3. Run application.

### 51.10.2 Guide SWO for Keil $\mu$ Vision

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. Make sure the SDK\_DEBUGCONSOLE\_UART is not defined, remove the default definition in fsl\_debug\_console.h.
2. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
3. Open Project>Options for target or using Alt+F7 or click.
4. Select "Debug" tab, select "J-Link/J-Trace Cortex" and click "Setting button".
5. Select "Debug" tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

#### Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

#### Step 4: Run the project

1. Choose "Debug" on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

### 51.10.3 Guide SWO for MCUXpresso IDE

**NOTE:** MCUX support SWO for LPC-Link2 debug probe only.

### 51.10.4 Guide SWO for ARMGCC

**NOTE:** ARMGCC has no library support SWO.



# Chapter 52

## Notification Framework

### 52.1 Overview

This section describes the programming interface of the Notifier driver.

### 52.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.  
The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *
    notify, void *data)
{
    status_t ret = kStatus_Success;

    ...
    ...
    ...

    return ret;
}

// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *userData)
```

## Notifier Overview

```
{
    ...
    ...
    ...
}
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
        kNOTIFIER_CallbackBeforeAfter,
        (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
        APP_PowerModeSwitch, NULL);
    ...
    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
        kNOTIFIER_PolicyAgreement);
}
```

## Data Structures

- struct [notifier\\_notification\\_block\\_t](#)  
*notification block passed to the registered callback function. [More...](#)*
- struct [notifier\\_callback\\_config\\_t](#)  
*Callback configuration structure. [More...](#)*
- struct [notifier\\_handle\\_t](#)  
*Notifier handle structure. [More...](#)*

## Typedefs

- typedef void [notifier\\_user\\_config\\_t](#)  
*Notifier user configuration type.*
- typedef status\_t(\* [notifier\\_user\\_function\\_t](#))([notifier\\_user\\_config\\_t](#) \*targetConfig, void \*userData)

- Notifier user function prototype Use this function to execute specific operations in configuration switch.
- typedef status\_t(\* notifier\_callback\_t)(notifier\_notification\_block\_t \*notify, void \*data)  
Callback prototype.

## Enumerations

- enum \_notifier\_status {  
kStatus\_NOTIFIER\_ErrorNotificationBefore,  
kStatus\_NOTIFIER\_ErrorNotificationAfter }  
Notifier error codes.
- enum notifier\_policy\_t {  
kNOTIFIER\_PolicyAgreement,  
kNOTIFIER\_PolicyForcible }  
Notifier policies.
- enum notifier\_notification\_type\_t {  
kNOTIFIER\_NotifyRecover = 0x00U,  
kNOTIFIER\_NotifyBefore = 0x01U,  
kNOTIFIER\_NotifyAfter = 0x02U }  
Notification type.
- enum notifier\_callback\_type\_t {  
kNOTIFIER\_CallbackBefore = 0x01U,  
kNOTIFIER\_CallbackAfter = 0x02U,  
kNOTIFIER\_CallbackBeforeAfter = 0x03U }  
The callback type, which indicates kinds of notification the callback handles.

## Functions

- status\_t NOTIFIER\_CreateHandle (notifier\_handle\_t \*notifierHandle, notifier\_user\_config\_t \*\*configs, uint8\_t configsNumber, notifier\_callback\_config\_t \*callbacks, uint8\_t callbacksNumber, notifier\_user\_function\_t userFunction, void \*userData)  
Creates a Notifier handle.
- status\_t NOTIFIER\_SwitchConfig (notifier\_handle\_t \*notifierHandle, uint8\_t configIndex, notifier\_policy\_t policy)  
Switches the configuration according to a pre-defined structure.
- uint8\_t NOTIFIER\_GetErrorCallbackIndex (notifier\_handle\_t \*notifierHandle)  
This function returns the last failed notification callback.

## 52.3 Data Structure Documentation

### 52.3.1 struct notifier\_notification\_block\_t

#### Data Fields

- notifier\_user\_config\_t \* targetConfig  
Pointer to target configuration.
- notifier\_policy\_t policy  
Configure transition policy.
- notifier\_notification\_type\_t notifyType

## Data Structure Documentation

*Configure notification type.*

### 52.3.1.0.0.64 Field Documentation

**52.3.1.0.0.64.1** `notifier_user_config_t* notifier_notification_block_t::targetConfig`

**52.3.1.0.0.64.2** `notifier_policy_t notifier_notification_block_t::policy`

**52.3.1.0.0.64.3** `notifier_notification_type_t notifier_notification_block_t::notifyType`

### 52.3.2 struct `notifier_callback_config_t`

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. `callback` - pointer to the callback function `callbackType` - specifies when the callback is called `callbackData` - pointer to the data passed to the callback.

#### Data Fields

- [`notifier\_callback\_t callback`](#)  
*Pointer to the callback function.*
- [`notifier\_callback\_type\_t callbackType`](#)  
*Callback type.*
- `void * callbackData`  
*Pointer to the data passed to the callback.*

### 52.3.2.0.0.65 Field Documentation

**52.3.2.0.0.65.1** `notifier_callback_t notifier_callback_config_t::callback`

**52.3.2.0.0.65.2** `notifier_callback_type_t notifier_callback_config_t::callbackType`

**52.3.2.0.0.65.3** `void* notifier_callback_config_t::callbackData`

### 52.3.3 struct `notifier_handle_t`

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. `NOTIFIER_CreateHandle()` must be called to initialize this handle.

#### Data Fields

- [`notifier\_user\_config\_t \*\* configsTable`](#)  
*Pointer to configure table.*
- `uint8_t configsNumber`  
*Number of configurations.*

- [notifier\\_callback\\_config\\_t](#) \* [callbacksTable](#)  
*Pointer to callback table.*
- [uint8\\_t](#) [callbacksNumber](#)  
*Maximum number of callback configurations.*
- [uint8\\_t](#) [errorCallbackIndex](#)  
*Index of callback returns error.*
- [uint8\\_t](#) [currentConfigIndex](#)  
*Index of current configuration.*
- [notifier\\_user\\_function\\_t](#) [userFunction](#)  
*User function.*
- [void](#) \* [userData](#)  
*User data passed to user function.*

#### 52.3.3.0.0.66 Field Documentation

52.3.3.0.0.66.1 [notifier\\_user\\_config\\_t](#)\*\* [notifier\\_handle\\_t::configsTable](#)

52.3.3.0.0.66.2 [uint8\\_t](#) [notifier\\_handle\\_t::configsNumber](#)

52.3.3.0.0.66.3 [notifier\\_callback\\_config\\_t](#)\* [notifier\\_handle\\_t::callbacksTable](#)

52.3.3.0.0.66.4 [uint8\\_t](#) [notifier\\_handle\\_t::callbacksNumber](#)

52.3.3.0.0.66.5 [uint8\\_t](#) [notifier\\_handle\\_t::errorCallbackIndex](#)

52.3.3.0.0.66.6 [uint8\\_t](#) [notifier\\_handle\\_t::currentConfigIndex](#)

52.3.3.0.0.66.7 [notifier\\_user\\_function\\_t](#) [notifier\\_handle\\_t::userFunction](#)

52.3.3.0.0.66.8 [void](#)\* [notifier\\_handle\\_t::userData](#)

## 52.4 Typedef Documentation

### 52.4.1 [typedef void notifier\\_user\\_config\\_t](#)

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

### 52.4.2 [typedef status\\_t\(\\* notifier\\_user\\_function\\_t\)\(notifier\\_user\\_config\\_t \\*targetConfig, void \\*userData\)](#)

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, [NOTIFIER\\_SwitchConfig\(\)](#) exits.

## Enumeration Type Documentation

### Parameters

<i>targetConfig</i>	target Configuration.
<i>userData</i>	Refers to other specific data passed to user function.

### Returns

An error code or `kStatus_Success`.

### 52.4.3 `typedef status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)`

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the `notifier_callback_config_t` callback configuration structure. Depending on callback type, function of this prototype is called (see `NOTIFIER_SwitchConfig()`) before configuration switch, after it or in both use cases to notify about the switch progress (see `notifier_callback_type_t`). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see `notifier_notification_block_t`) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see `notifier_policy_t`), the callback may deny the execution of the user function by returning an error code different than `kStatus_Success` (see `NOTIFIER_SwitchConfig()`).

### Parameters

<i>notify</i>	Notification block.
<i>data</i>	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

### Returns

An error code or `kStatus_Success`.

## 52.5 Enumeration Type Documentation

### 52.5.1 `enum _notifier_status`

Used as return value of Notifier functions.

### Enumerator

***kStatus\_NOTIFIER\_ErrorNotificationBefore*** An error occurs during send "BEFORE" notification.

***kStatus\_NOTIFIER\_ErrorNotificationAfter*** An error occurs during send "AFTER" notification.

### 52.5.2 enum notifier\_policy\_t

Defines whether the user function execution is forced or not. For `kNOTIFIER_PolicyForcible`, the user function is executed regardless of the callback results, while `kNOTIFIER_PolicyAgreement` policy is used to exit `NOTIFIER_SwitchConfig()` when any of the callbacks returns error code. See also `NOTIFIER_SwitchConfig()` description.

Enumerator

***kNOTIFIER\_PolicyAgreement*** `NOTIFIER_SwitchConfig()` method is exited when any of the callbacks returns error code.

***kNOTIFIER\_PolicyForcible*** The user function is executed regardless of the results.

### 52.5.3 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.

### 52.5.4 enum notifier\_callback\_type\_t

Used in the callback configuration structure (`notifier_callback_config_t`) to specify when the registered callback is called during configuration switch initiated by the `NOTIFIER_SwitchConfig()`. Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect `NOTIFIER_SwitchConfig()` execution. See the `NOTIFIER_SwitchConfig()` and `notifier_policy_t` documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

***kNOTIFIER\_CallbackBefore*** Callback handles BEFORE notification.

***kNOTIFIER\_CallbackAfter*** Callback handles AFTER notification.

***kNOTIFIER\_CallbackBeforeAfter*** Callback handles BEFORE and AFTER notification.

### 52.6 Function Documentation

**52.6.1** `status_t NOTIFIER_CreateHandle ( notifier_handle_t * notifierHandle,  
notifier_user_config_t ** configs, uint8_t configsNumber, notifier_callback-  
_config_t * callbacks, uint8_t callbacksNumber, notifier_user_function_t  
userFunction, void * userData )`



## Parameters

<i>notifierHandle</i>	A pointer to the notifier handle.
<i>configs</i>	A pointer to an array with references to all configurations which is handled by the Notifier.
<i>configsNumber</i>	Number of configurations. Size of the configuration array.
<i>callbacks</i>	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
<i>callbacks-Number</i>	Number of registered callbacks. Size of the callbacks array.
<i>userFunction</i>	User function.
<i>userData</i>	User data passed to user function.

## Returns

An error Code or kStatus\_Success.

### 52.6.2 **status\_t NOTIFIER\_SwitchConfig ( notifier\_handle\_t \* *notifierHandle*, uint8\_t *configIndex*, notifier\_policy\_t *policy* )**

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER\_PolicyForcible) or exited (kNOTIFIER\_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER\_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when [NOTIFIER\\_SwitchConfig\(\)](#) exits.

## Parameters

## Function Documentation

<i>notifierHandle</i>	pointer to notifier handle
<i>configIndex</i>	Index of the target configuration.
<i>policy</i>	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus\_Success.

### 52.6.3 uint8\_t NOTIFIER\_GetErrorCallbackIndex ( notifier\_handle\_t \* *notifierHandle* )

This function returns an index of the last callback that failed during the configuration switch while the last [NOTIFIER\\_SwitchConfig\(\)](#) was called. If the last [NOTIFIER\\_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

<i>notifierHandle</i>	Pointer to the notifier handle
-----------------------	--------------------------------

Returns

Callback Index of the last failed callback or value equal to callbacks count.

# Chapter 53

## Shell

### 53.1 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.

### 53.2 Function groups

#### 53.2.1 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>> ");
```

#### 53.2.2 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.

### 53.2.3 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](#) (20U)  
*Macro to set maximum count of commands.*
- #define [SHELL\\_OPTIONAL\\_PARAMS](#) (0xFF)  
*Macro to bypass arguments check.*

## 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 configures 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.*

## 53.3 Data Structure Documentation

### 53.3.1 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](#)

## Data Structure Documentation

- *Total history command in hist buff.*  
char `hist_buf` [SHELL\_HIST\_MAX][SHELL\_BUFFER\_SIZE]
- *History buffer.*  
bool `exit`  
*Exit Flag.*

### 53.3.2 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.*

#### 53.3.2.0.0.67 Field Documentation

##### 53.3.2.0.0.67.1 const char\* shell\_command\_context\_t::pcCommand

For example "help". It must be all lower case.

##### 53.3.2.0.0.67.2 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".

##### 53.3.2.0.0.67.3 const cmd\_function\_t shell\_command\_context\_t::pFuncCallBack

##### 53.3.2.0.0.67.4 uint8\_t shell\_command\_context\_t::cExpectedNumberOfParameters

### 53.3.3 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.*

## 53.4 Macro Definition Documentation

53.4.1 `#define SHELL_USE_HISTORY (0U)`

53.4.2 `#define SHELL_SEARCH_IN_HIST (1U)`

53.4.3 `#define SHELL_USE_FILE_STREAM (0U)`

53.4.4 `#define SHELL_AUTO_COMPLETE (1U)`

53.4.5 `#define SHELL_BUFFER_SIZE (64U)`

53.4.6 `#define SHELL_MAX_ARGS (8U)`

53.4.7 `#define SHELL_HIST_MAX (3U)`

53.4.8 `#define SHELL_MAX_CMD (20U)`

## 53.5 Typedef Documentation

53.5.1 `typedef void(* send_data_cb_t)(uint8_t *buf, uint32_t len)`

53.5.2 `typedef void(* recv_data_cb_t)(uint8_t *buf, uint32_t len)`

53.5.3 `typedef int(* printf_data_t)(const char *format,...)`

53.5.4 `typedef int32_t(* cmd_function_t)(p_shell_context_t context, int32_t argc, char **argv)`

## 53.6 Enumeration Type Documentation

53.6.1 `enum fun_key_status_t`

Enumerator

*kSHELL\_Normal* Normal key.

*kSHELL\_Special* Special key.

*kSHELL\_Function* Function key.

## Function Documentation

### 53.7 Function Documentation

#### 53.7.1 void SHELL\_Init ( p\_shell\_context\_t context, send\_data\_cb\_t send\_cb, rcv\_data\_cb\_t rcv\_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. This is an 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>rcv_cb</i>	The pointer to call back receive data function.
<i>prompt</i>	The string prompt of Shell

#### 53.7.2 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

#### 53.7.3 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.



## Chapter 54

### Flexio\_mculcd\_edma

#### 54.1 Overview

##### Data Structures

- struct [flexio\\_mculcd\\_edma\\_handle\\_t](#)  
*FlexIO MCULCD eDMA transfer handle, users should not touch the content of the handle. [More...](#)*

##### Macros

- #define [FSL\\_FLEXIO\\_MCULCD\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 2))  
*FlexIO MCULCD EDMA driver version 2.0.2.*

##### Typedefs

- typedef void(\* [flexio\\_mculcd\\_edma\\_transfer\\_callback\\_t](#))([FLEXIO\\_MCULCD\\_Type](#) \*base, [flexio\\_mculcd\\_edma\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
*FlexIO MCULCD master callback for transfer complete.*

##### eDMA Transactional

- [status\\_t FLEXIO\\_MCULCD\\_TransferCreateHandleEDMA](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, [flexio\\_mculcd\\_edma\\_handle\\_t](#) \*handle, [flexio\\_mculcd\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*txEdmaHandle, [edma\\_handle\\_t](#) \*rxEdmaHandle)  
*Initializes the FLEXIO MCULCD master eDMA handle.*
- [status\\_t FLEXIO\\_MCULCD\\_TransferEDMA](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, [flexio\\_mculcd\\_edma\\_handle\\_t](#) \*handle, [flexio\\_mculcd\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking FlexIO MCULCD transfer using eDMA.*
- [void FLEXIO\\_MCULCD\\_TransferAbortEDMA](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, [flexio\\_mculcd\\_edma\\_handle\\_t](#) \*handle)  
*Aborts a FlexIO MCULCD transfer using eDMA.*
- [status\\_t FLEXIO\\_MCULCD\\_TransferGetCountEDMA](#) ([FLEXIO\\_MCULCD\\_Type](#) \*base, [flexio\\_mculcd\\_edma\\_handle\\_t](#) \*handle, [size\\_t](#) \*count)  
*Gets the remaining bytes for FlexIO MCULCD eDMA transfer.*

#### 54.2 Data Structure Documentation

##### 54.2.1 struct\_flexio\_mculcd\_edma\_handle

typedef for [flexio\\_mculcd\\_edma\\_handle\\_t](#) in advance.

## Data Structure Documentation

### Data Fields

- [FLEXIO\\_MCULCD\\_Type](#) \* base  
*Pointer to the [FLEXIO\\_MCULCD\\_Type](#).*
- `uint8_t txShifterNum`  
*Number of shifters used for TX.*
- `uint8_t rxShifterNum`  
*Number of shifters used for RX.*
- `uint32_t minorLoopBytes`  
*eDMA transfer minor loop bytes.*
- `edma_modulo_t txEdmaModulo`  
*Modulo value for the FlexIO shifter buffer access.*
- `edma_modulo_t rxEdmaModulo`  
*Modulo value for the FlexIO shifter buffer access.*
- `uint32_t dataAddrOrSameValue`  
*When sending the same value for many times, this is the value to send.*
- `size_t dataCount`  
*Total count to be transferred.*
- `volatile size_t remainingCount`  
*Remaining count still not transferred.*
- `volatile uint32_t state`  
*FlexIO MCULCD driver internal state.*
- `edma_handle_t * txEdmaHandle`  
*DMA handle for MCULCD TX.*
- `edma_handle_t * rxEdmaHandle`  
*DMA handle for MCULCD RX.*
- `flexio_mculcd_edma_transfer_callback_t completionCallback`  
*Callback for MCULCD DMA transfer.*
- `void * userData`  
*User Data for MCULCD DMA callback.*

#### 54.2.1.0.0.68 Field Documentation

54.2.1.0.0.68.1 `FLEXIO_MCULCD_Type* flexio_mculcd_edma_handle_t::base`

54.2.1.0.0.68.2 `uint8_t flexio_mculcd_edma_handle_t::txShifterNum`

54.2.1.0.0.68.3 `uint8_t flexio_mculcd_edma_handle_t::rxShifterNum`

54.2.1.0.0.68.4 `uint32_t flexio_mculcd_edma_handle_t::minorLoopBytes`

54.2.1.0.0.68.5 `edma_modulo_t flexio_mculcd_edma_handle_t::txEdmaModulo`

54.2.1.0.0.68.6 `edma_modulo_t flexio_mculcd_edma_handle_t::rxEdmaModulo`

54.2.1.0.0.68.7 `uint32_t flexio_mculcd_edma_handle_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

54.2.1.0.0.68.8 `size_t flexio_mculcd_edma_handle_t::dataCount`

54.2.1.0.0.68.9 `volatile size_t flexio_mculcd_edma_handle_t::remainingCount`

54.2.1.0.0.68.10 `volatile uint32_t flexio_mculcd_edma_handle_t::state`

### 54.3 Macro Definition Documentation

54.3.1 `#define FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

### 54.4 Typedef Documentation

54.4.1 `typedef void(* flexio_mculcd_edma_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculcd_edma_handle_t *handle, status_t status, void *userData)`

When transfer finished, the callback function is called and returns the `status` as `kStatus_FLEXIO_MCULCD_Idle`.

### 54.5 Function Documentation

54.5.1 `status_t FLEXIO_MCULCD_TransferCreateHandleEDMA ( FLEXIO_MCULCD_Type * base, flexio_mculcd_edma_handle_t * handle, flexio_mculcd_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`

This function initializes the FLEXIO MCULCD master eDMA handle which can be used for other FLEXIO MCULCD transactional APIs. For a specified FLEXIO MCULCD instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	Pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.
<i>handle</i>	Pointer to <code>flexio_mculcd_edma_handle_t</code> structure to store the transfer state.
<i>callback</i>	MCULCD transfer complete callback, NULL means no callback.
<i>userData</i>	callback function parameter.
<i>txEdmaHandle</i>	User requested eDMA handle for FlexIO MCULCD eDMA TX, the DMA request source of this handle should be the first of TX shifters.

## Function Documentation

<i>rxEdmaHandle</i>	User requested eDMA handle for FlexIO MCULCD eDMA RX, the DMA request source of this handle should be the last of RX shifters.
---------------------	--------------------------------------------------------------------------------------------------------------------------------

Return values

<i>kStatus_Success</i>	Successfully create the handle.
------------------------	---------------------------------

### 54.5.2 **status\_t FLEXIO\_MCULCD\_TransferEDMA ( FLEXIO\_MCULCD\_Type \* base, flexio\_mculcd\_edma\_handle\_t \* handle, flexio\_mculcd\_transfer\_t \* xfer )**

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

1. Use the transfer completed callback;
2. Polling function FLEXIO\_MCULCD\_GetTransferCountEDMA

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.
<i>handle</i>	pointer to flexio_mculcd_edma_handle_t structure to store the transfer state.
<i>xfer</i>	Pointer to FlexIO MCULCD transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_FLEXIO_MCULCD_Busy</i>	FlexIO MCULCD is not idle, it is running another transfer.

### 54.5.3 **void FLEXIO\_MCULCD\_TransferAbortEDMA ( FLEXIO\_MCULCD\_Type \* base, flexio\_mculcd\_edma\_handle\_t \* handle )**

Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.
<i>handle</i>	FlexIO MCULCD eDMA handle pointer.

**54.5.4 status\_t FLEXIO\_MCULCD\_TransferGetCountEDMA ( FLEXIO\_MCULCD\_Type \* *base*, flexio\_mculcd\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

## Parameters

<i>base</i>	pointer to <a href="#">FLEXIO_MCULCD_Type</a> structure.
<i>handle</i>	FlexIO MCULCD eDMA handle pointer.
<i>count</i>	Number of count transferred so far by the eDMA transaction.

## Return values

<i>kStatus_Success</i>	Get the transferred count Successfully.
<i>kStatus_NoTransferInProgress</i>	No tranfer in process.





## Chapter 55

### Data Structure Documentation

#### 55.0.5 `bee_region_config_t` Struct Reference

BEE region configuration structure.

```
#include <fsl_bee.h>
```

##### Data Fields

- `bee_aes_mode_t` `mode`  
*AES mode used for encryption/decryption.*
- `uint32_t` `regionBot`  
*Region bottom address.*
- `uint32_t` `regionTop`  
*Region top address.*
- `uint32_t` `addrOffset`  
*Region address offset.*
- `bee_region_enable_t` `regionEn`  
*Region enable/disable.*

##### 55.0.5.1 Detailed Description



**How to Reach Us:**

**Home Page:**

[nxp.com](http://nxp.com)

**Web Support:**

[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

NXP reserves the right to make changes without further notice to any products herein. NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

NXP, the NXP logo, Freescale, the Freescale logo, Kinetis, Processor Expert are trademarks of NXP B.V. Tower is a trademark of NXP. All other product or service names are the property of their respective owners. Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2018 NXP B.V.

Document Number: MCUXSDKIMXRT1050APIRM

Rev. 0

May 2018

The logo for Arm, consisting of the word "arm" in a lowercase, blue, sans-serif font.