

# MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKIMX6ULAPIRM  
Rev. 0  
Mar 2017



Confidential Proprietary

# 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>11</b>
<b>5.2</b>	<b>Typical use case</b>	<b>11</b>
5.2.1	Polling Configuration	11
5.2.2	Polling Configuration	12
<b>5.3</b>	<b>Data Structure Documentation</b>	<b>15</b>
5.3.1	struct adc_config_t	15
5.3.2	struct adc_offset_config_t	16
5.3.3	struct adc_hardware_compare_config_t	16
5.3.4	struct adc_channel_config_t	16
<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>17</b>
5.5.1	adc_status_flags_t	17
5.5.2	adc_reference_voltage_source_t	17
5.5.3	adc_sample_period_mode_t	17
5.5.4	adc_clock_source_t	17
5.5.5	adc_clock_driver_t	18
5.5.6	adc_resolution_t	18
5.5.7	adc_hardware_compare_mode_t	18
5.5.8	adc_hardware_average_mode_t	18
<b>5.6</b>	<b>Function Documentation</b>	<b>19</b>
5.6.1	ADC_Init	19
5.6.2	ADC_Deinit	19

# Contents

Section Number	Title	Page Number
5.6.3	ADC_GetDefaultConfig . . . . .	19
5.6.4	ADC_SetChannelConfig . . . . .	20
5.6.5	ADC_GetChannelConversionValue . . . . .	20
5.6.6	ADC_GetChannelStatusFlags . . . . .	21
5.6.7	ADC_DoAutoCalibration . . . . .	21
5.6.8	ADC_SetOffsetConfig . . . . .	22
5.6.9	ADC_EnableDMA . . . . .	22
5.6.10	ADC_EnableHardwareTrigger . . . . .	22
5.6.11	ADC_SetHardwareCompareConfig . . . . .	22
5.6.12	ADC_SetHardwareAverageConfig . . . . .	23
5.6.13	ADC_GetStatusFlags . . . . .	23
5.6.14	ADC_ClearStatusFlags . . . . .	23
<b>5.7</b>	<b>Variable Documentation</b> . . . . .	<b>24</b>
5.7.1	enableOverWrite . . . . .	24
5.7.2	enableContinuousConversion . . . . .	24
5.7.3	enableHighSpeed . . . . .	24
5.7.4	enableLowPower . . . . .	24
5.7.5	enableLongSample . . . . .	24
5.7.6	enableAsynchronousClockOutput . . . . .	24
5.7.7	referenceVoltageSource . . . . .	24
5.7.8	samplePeriodMode . . . . .	24
5.7.9	clockSource . . . . .	24
5.7.10	clockDriver . . . . .	24
5.7.11	resolution . . . . .	24
5.7.12	enableSigned . . . . .	24
5.7.13	offsetValue . . . . .	24
5.7.14	hardwareCompareMode . . . . .	24
5.7.15	value1 . . . . .	25
5.7.16	value2 . . . . .	25
5.7.17	channelNumber . . . . .	25
5.7.18	enableInterruptOnConversionCompleted . . . . .	25
<b>Chapter ADC_5HC: 12-bit Analog to Digital Converter Driver</b>		
<b>6.1</b>	<b>Overview</b> . . . . .	<b>27</b>
<b>6.2</b>	<b>Typical use case</b> . . . . .	<b>27</b>
6.2.1	Polling Configuration . . . . .	27
6.2.2	Polling Configuration . . . . .	28
<b>6.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>31</b>
6.3.1	struct adc_5hc_config_t . . . . .	31
6.3.2	struct adc_5hc_offset_config_t . . . . .	32
6.3.3	struct adc_5hc_hardware_compare_config_t . . . . .	32

Section Number	Title	Page Number
6.3.4	<code>struct adc_5hc_channel_config_t</code>	32
<b>6.4</b>	<b>Macro Definition Documentation</b>	<b>33</b>
6.4.1	<code>FSL_ADC_5HC_DRIVER_VERSION</code>	33
<b>6.5</b>	<b>Enumeration Type Documentation</b>	<b>33</b>
6.5.1	<code>adc_5hc_status_flags_t</code>	33
6.5.2	<code>adc_5hc_reference_voltage_source_t</code>	33
6.5.3	<code>adc_5hc_sample_period_mode_t</code>	33
6.5.4	<code>adc_5hc_clock_source_t</code>	34
6.5.5	<code>adc_5hc_clock_driver_t</code>	34
6.5.6	<code>adc_5hc_resolution_t</code>	34
6.5.7	<code>adc_5hc_hardware_compare_mode_t</code>	34
6.5.8	<code>adc_5hc_hardware_average_mode_t</code>	35
<b>6.6</b>	<b>Function Documentation</b>	<b>35</b>
6.6.1	<code>ADC_5HC_Init</code>	35
6.6.2	<code>ADC_5HC_Deinit</code>	35
6.6.3	<code>ADC_5HC_GetDefaultConfig</code>	35
6.6.4	<code>ADC_5HC_SetChannelConfig</code>	36
6.6.5	<code>ADC_5HC_GetChannelConversionValue</code>	36
6.6.6	<code>ADC_5HC_GetChannelStatusFlags</code>	37
6.6.7	<code>ADC_5HC_DoAutoCalibration</code>	37
6.6.8	<code>ADC_5HC_SetOffsetConfig</code>	38
6.6.9	<code>ADC_5HC_EnableDMA</code>	38
6.6.10	<code>ADC_5HC_EnableHardwareTrigger</code>	38
6.6.11	<code>ADC_5HC_SetHardwareCompareConfig</code>	38
6.6.12	<code>ADC_5HC_SetHardwareAverageConfig</code>	39
6.6.13	<code>ADC_5HC_GetStatusFlags</code>	39
6.6.14	<code>ADC_5HC_ClearStatusFlags</code>	39
<b>6.7</b>	<b>Variable Documentation</b>	<b>40</b>
6.7.1	<code>enableOverWrite</code>	40
6.7.2	<code>enableContinuousConversion</code>	40
6.7.3	<code>enableHighSpeed</code>	40
6.7.4	<code>enableLowPower</code>	40
6.7.5	<code>enableLongSample</code>	40
6.7.6	<code>enableAsynchronousClockOutput</code>	40
6.7.7	<code>referenceVoltageSource</code>	40
6.7.8	<code>samplePeriodMode</code>	40
6.7.9	<code>clockSource</code>	40
6.7.10	<code>clockDriver</code>	40
6.7.11	<code>resolution</code>	40
6.7.12	<code>enableSigned</code>	40
6.7.13	<code>offsetValue</code>	40

# Contents

Section Number	Title	Page Number
6.7.14	hardwareCompareMode . . . . .	40
6.7.15	value1 . . . . .	41
6.7.16	value2 . . . . .	41
6.7.17	channelNumber . . . . .	41
6.7.18	enableInterruptOnConversionCompleted . . . . .	41

## Chapter CACHE: CACHE Memory Controller

7.1	<b>Overview</b> . . . . .	43
7.2	<b>Function groups</b> . . . . .	43
7.3	<b>Macro Definition Documentation</b> . . . . .	44
7.3.1	FSL_CACHE_DRIVER_VERSION . . . . .	44
7.4	<b>Function Documentation</b> . . . . .	44
7.4.1	L1CACHE_InvalidateICacheByRange . . . . .	44
7.4.2	L1CACHE_InvalidateDCacheByRange . . . . .	45
7.4.3	L1CACHE_CleanDCacheByRange . . . . .	46
7.4.4	L1CACHE_CleanInvalidateDCacheByRange . . . . .	46
7.4.5	ICACHE_InvalidateByRange . . . . .	47
7.4.6	DCACHE_InvalidateByRange . . . . .	48
7.4.7	DCACHE_CleanByRange . . . . .	48
7.4.8	DCACHE_CleanInvalidateByRange . . . . .	49

## Chapter CSI: CMOS Sensor Interface

8.1	<b>Overview</b> . . . . .	51
8.2	<b>Frame Buffer Queue</b> . . . . .	51
8.3	<b>Typical use case</b> . . . . .	51
8.3.1	Receive with functional APIs . . . . .	51
8.3.2	Receive with transactional APIs . . . . .	52
8.4	<b>Data Structure Documentation</b> . . . . .	56
8.4.1	struct csi_config_t . . . . .	56
8.4.2	struct _csi_handle . . . . .	57
8.5	<b>Macro Definition Documentation</b> . . . . .	58
8.5.1	CSI_DRIVER_QUEUE_SIZE . . . . .	58
8.6	<b>Typedef Documentation</b> . . . . .	58
8.6.1	csi_transfer_callback_t . . . . .	58
8.7	<b>Enumeration Type Documentation</b> . . . . .	59

# Contents

Section Number	Title	Page Number
8.7.1	_csi_status . . . . .	59
8.7.2	csi_work_mode_t . . . . .	59
8.7.3	csi_data_bus_t . . . . .	59
8.7.4	_csi_polarity_flags . . . . .	59
8.7.5	csi_fifo_t . . . . .	60
8.7.6	_csi_interrupt_enable . . . . .	60
8.7.7	_csi_flags . . . . .	60
<b>8.8</b>	<b>Function Documentation</b> . . . . .	<b>61</b>
8.8.1	CSI_Init . . . . .	61
8.8.2	CSI_Deinit . . . . .	61
8.8.3	CSI_Reset . . . . .	62
8.8.4	CSI_GetDefaultConfig . . . . .	62
8.8.5	CSI_ClearFifo . . . . .	62
8.8.6	CSI_ReflashFifoDma . . . . .	62
8.8.7	CSI_EnableFifoDmaRequest . . . . .	63
8.8.8	CSI_Start . . . . .	63
8.8.9	CSI_Stop . . . . .	63
8.8.10	CSI_SetRxBufferAddr . . . . .	63
8.8.11	CSI_EnableInterrupts . . . . .	64
8.8.12	CSI_DisableInterrupts . . . . .	64
8.8.13	CSI_GetStatusFlags . . . . .	64
8.8.14	CSI_ClearStatusFlags . . . . .	64
8.8.15	CSI_TransferCreateHandle . . . . .	65
8.8.16	CSI_TransferStart . . . . .	65
8.8.17	CSI_TransferStop . . . . .	66
8.8.18	CSI_TransferSubmitEmptyBuffer . . . . .	66
8.8.19	CSI_TransferGetFullBuffer . . . . .	67
8.8.20	CSI_TransferHandleIRQ . . . . .	67

## Chapter ECSPI: Serial Peripheral Interface Driver

<b>9.1</b>	<b>Overview</b> . . . . .	<b>69</b>
<b>9.2</b>	<b>ECSPI Driver</b> . . . . .	<b>70</b>
9.2.1	Overview . . . . .	70
9.2.2	Typical use case . . . . .	70
9.2.3	Data Structure Documentation . . . . .	76
9.2.4	Macro Definition Documentation . . . . .	78
9.2.5	Enumeration Type Documentation . . . . .	78
9.2.6	Function Documentation . . . . .	81
<b>9.3</b>	<b>ECSPI FreeRTOS Driver</b> . . . . .	<b>93</b>
9.3.1	Overview . . . . .	93
9.3.2	Function Documentation . . . . .	93

# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>eLCDIF: Enhanced LCD Interface</b>	
<b>10.1</b>	<b>Overview</b>	<b>95</b>
<b>10.2</b>	<b>Typical use case</b>	<b>95</b>
10.2.1	Frame buffer update	95
10.2.2	Alpha surface	96
<b>10.3</b>	<b>Data Structure Documentation</b>	<b>100</b>
10.3.1	struct elcdif_pixel_format_reg_t	100
10.3.2	struct elcdif_rgb_mode_config_t	100
10.3.3	struct elcdif_as_buffer_config_t	101
10.3.4	struct elcdif_as_blend_config_t	102
<b>10.4</b>	<b>Macro Definition Documentation</b>	<b>102</b>
10.4.1	FSL_ELCDIF_DRIVER_VERSION	102
<b>10.5</b>	<b>Enumeration Type Documentation</b>	<b>102</b>
10.5.1	_elcdif_polarity_flags	102
10.5.2	_elcdif_interrupt_enable	103
10.5.3	_elcdif_interrupt_flags	103
10.5.4	_elcdif_status_flags	103
10.5.5	elcdif_pixel_format_t	104
10.5.6	elcdif_lcd_data_bus_t	104
10.5.7	elcdif_as_pixel_format_t	104
10.5.8	elcdif_alpha_mode_t	104
10.5.9	elcdif_rop_mode_t	105
<b>10.6</b>	<b>Function Documentation</b>	<b>105</b>
10.6.1	ELCDIF_RgbModeInit	105
10.6.2	ELCDIF_GetStatus	105
10.6.3	ELCDIF_GetLIFOCount	107
10.6.4	ELCDIF_EnableInterrupts	107
10.6.5	ELCDIF_DisableInterrupts	107
10.6.6	ELCDIF_GetInterruptStatus	108
10.6.7	ELCDIF_ClearInterruptStatus	108
10.6.8	ELCDIF_SetAlphaSurfaceBufferConfig	108
10.6.9	ELCDIF_SetAlphaSurfaceBlendConfig	108
10.6.10	ELCDIF_SetNextAlphaSurfaceBufferAddr	109
10.6.11	ELCDIF_SetOverlayColorKey	109
10.6.12	ELCDIF_EnableOverlayColorKey	109
10.6.13	ELCDIF_EnableAlphaSurface	110
10.6.14	ELCDIF_EnableProcessSurface	111

# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>ENET: Ethernet MAC Driver</b>	
<b>11.1</b>	<b>Overview</b>	<b>113</b>
<b>11.2</b>	<b>Typical use case</b>	<b>113</b>
11.2.1	ENET Initialization, receive, and transmit operations	113
<b>11.3</b>	<b>Data Structure Documentation</b>	<b>121</b>
11.3.1	struct enet_rx_bd_struct_t	121
11.3.2	struct enet_tx_bd_struct_t	122
11.3.3	struct enet_data_error_stats_t	122
11.3.4	struct enet_buffer_config_t	123
11.3.5	struct enet_intcoalesce_config_t	124
11.3.6	struct enet_config_t	124
11.3.7	struct _enet_handle	126
<b>11.4</b>	<b>Macro Definition Documentation</b>	<b>127</b>
11.4.1	FSL_ENET_DRIVER_VERSION	127
11.4.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	129
11.4.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	129
11.4.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	129
11.4.5	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	129
11.4.6	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	129
11.4.7	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	129
11.4.8	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	129
11.4.9	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	129
11.4.10	ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK	129
11.4.11	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	129
11.4.12	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	129
11.4.13	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	129
11.4.14	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	129
11.4.15	ENET_BUFFDESCRIPTOR_TX_READY_MASK	129
11.4.16	ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK	129
11.4.17	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	129
11.4.18	ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK	129
11.4.19	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	129
11.4.20	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	129
11.4.21	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	129
11.4.22	ENET_FRAME_MAX_FRAMELEN	130
11.4.23	ENET_FIFO_MIN_RX_FULL	130
11.4.24	ENET_RX_MIN_BUFFERSIZE	130
11.4.25	ENET_PHY_MAXADDRESS	130
<b>11.5</b>	<b>Typedef Documentation</b>	<b>130</b>
11.5.1	enet_callback_t	130

# Contents

Section Number	Title	Page Number
<b>11.6</b>	<b>Enumeration Type Documentation</b>	<b>130</b>
11.6.1	_enet_status . . . . .	130
11.6.2	enet_mii_mode_t . . . . .	130
11.6.3	enet_mii_speed_t . . . . .	130
11.6.4	enet_mii_duplex_t . . . . .	131
11.6.5	enet_mii_write_t . . . . .	131
11.6.6	enet_mii_read_t . . . . .	131
11.6.7	enet_mii_extend_opcode . . . . .	131
11.6.8	enet_special_control_flag_t . . . . .	131
11.6.9	enet_interrupt_enable_t . . . . .	132
11.6.10	enet_event_t . . . . .	132
11.6.11	enet_tx_accelerator_t . . . . .	133
11.6.12	enet_rx_accelerator_t . . . . .	133
<b>11.7</b>	<b>Function Documentation</b>	<b>133</b>
11.7.1	ENET_GetDefaultConfig . . . . .	133
11.7.2	ENET_Init . . . . .	133
11.7.3	ENET_Deinit . . . . .	134
11.7.4	ENET_Reset . . . . .	134
11.7.5	ENET_SetMII . . . . .	134
11.7.6	ENET_SetSMI . . . . .	135
11.7.7	ENET_GetSMI . . . . .	135
11.7.8	ENET_ReadSMIData . . . . .	135
11.7.9	ENET_StartSMIRead . . . . .	136
11.7.10	ENET_StartSMIWrite . . . . .	136
11.7.11	ENET_StartExtC45SMIRead . . . . .	136
11.7.12	ENET_StartExtC45SMIWrite . . . . .	137
11.7.13	ENET_SetMacAddr . . . . .	137
11.7.14	ENET_GetMacAddr . . . . .	137
11.7.15	ENET_AddMulticastGroup . . . . .	138
11.7.16	ENET_LeaveMulticastGroup . . . . .	139
11.7.17	ENET_ActiveRead . . . . .	139
11.7.18	ENET_EnableSleepMode . . . . .	139
11.7.19	ENET_GetAccelFunction . . . . .	139
11.7.20	ENET_EnableInterrupts . . . . .	140
11.7.21	ENET_DisableInterrupts . . . . .	140
11.7.22	ENET_GetInterruptStatus . . . . .	141
11.7.23	ENET_ClearInterruptStatus . . . . .	141
11.7.24	ENET_SetCallback . . . . .	141
11.7.25	ENET_GetRxErrBeforeReadFrame . . . . .	142
11.7.26	ENET_GetRxFrameSize . . . . .	142
11.7.27	ENET_ReadFrame . . . . .	143
11.7.28	ENET_SendFrame . . . . .	144
11.7.29	ENET_TransmitIRQHandler . . . . .	144
11.7.30	ENET_ReceiveIRQHandler . . . . .	145

# Contents

Section Number	Title	Page Number
11.7.31	ENET_ErrorIRQHandler . . . . .	145
11.7.32	ENET_CommonFrame0IRQHandler . . . . .	145

## Chapter EPI: Enhanced Periodic Interrupt Timer

12.1	Overview . . . . .	147
12.2	Function groups . . . . .	147
12.2.1	Initialization and deinitialization . . . . .	147
12.3	Typical use case . . . . .	147
12.3.1	EPI interrupt example . . . . .	147
12.4	Data Structure Documentation . . . . .	150
12.4.1	struct epit_config_t . . . . .	150
12.5	Enumeration Type Documentation . . . . .	151
12.5.1	epit_clock_source_t . . . . .	151
12.5.2	epit_output_operation_mode_t . . . . .	151
12.5.3	epit_interrupt_enable_t . . . . .	151
12.5.4	epit_status_flags_t . . . . .	151
12.6	Function Documentation . . . . .	151
12.6.1	EPI_SoftwareReset . . . . .	151
12.6.2	EPI_Init . . . . .	152
12.6.3	EPI_Deinit . . . . .	152
12.6.4	EPI_GetDefaultConfig . . . . .	152
12.6.5	EPI_SetClockSource . . . . .	153
12.6.6	EPI_SetClockDivider . . . . .	153
12.6.7	EPI_GetClockDivider . . . . .	153
12.6.8	EPI_StartTimer . . . . .	153
12.6.9	EPI_StopTimer . . . . .	154
12.6.10	EPI_SetTimerPeriod . . . . .	154
12.6.11	EPI_GetCurrentTimerCount . . . . .	154
12.6.12	EPI_SetOutputOperationMode . . . . .	155
12.6.13	EPI_SetOutputCompareValue . . . . .	155
12.6.14	EPI_EnableInterrupts . . . . .	155
12.6.15	EPI_DisableInterrupts . . . . .	155
12.6.16	EPI_GetEnabledInterrupts . . . . .	156
12.6.17	EPI_GetStatusFlags . . . . .	156
12.6.18	EPI_ClearStatusFlags . . . . .	156

## Chapter FlexCAN: Flex Controller Area Network Driver

13.1	Overview . . . . .	157
------	--------------------	-----

# Contents

Section Number	Title	Page Number
<b>13.2</b>	<b>FlexCAN Driver</b>	<b>158</b>
13.2.1	Overview . . . . .	158
13.2.2	Typical use case . . . . .	158
13.2.3	Data Structure Documentation . . . . .	166
13.2.4	Macro Definition Documentation . . . . .	170
13.2.5	Typedef Documentation . . . . .	175
13.2.6	Enumeration Type Documentation . . . . .	175
13.2.7	Function Documentation . . . . .	178
<b>13.3</b>	<b>FlexCAN eDMA Driver</b>	<b>193</b>
13.3.1	Overview . . . . .	193
13.3.2	Data Structure Documentation . . . . .	193
13.3.3	Typedef Documentation . . . . .	194
13.3.4	Function Documentation . . . . .	194
 <b>Chapter GPT: General Purpose Timer</b>		
<b>14.1</b>	<b>Overview</b> . . . . .	<b>197</b>
<b>14.2</b>	<b>Function groups</b> . . . . .	<b>197</b>
14.2.1	Initialization and deinitialization . . . . .	197
<b>14.3</b>	<b>Typical use case</b> . . . . .	<b>197</b>
14.3.1	GPT interrupt example . . . . .	197
<b>14.4</b>	<b>Data Structure Documentation</b> . . . . .	<b>201</b>
14.4.1	struct gpt_config_t . . . . .	201
<b>14.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>202</b>
14.5.1	gpt_clock_source_t . . . . .	202
14.5.2	gpt_input_capture_channel_t . . . . .	202
14.5.3	gpt_input_operation_mode_t . . . . .	202
14.5.4	gpt_output_compare_channel_t . . . . .	203
14.5.5	gpt_output_operation_mode_t . . . . .	203
14.5.6	gpt_interrupt_enable_t . . . . .	203
14.5.7	gpt_status_flag_t . . . . .	203
<b>14.6</b>	<b>Function Documentation</b> . . . . .	<b>204</b>
14.6.1	GPT_Init . . . . .	204
14.6.2	GPT_Deinit . . . . .	204
14.6.3	GPT_GetDefaultConfig . . . . .	204
14.6.4	GPT_SoftwareReset . . . . .	204
14.6.5	GPT_SetClockSource . . . . .	205
14.6.6	GPT_GetClockSource . . . . .	205
14.6.7	GPT_SetClockDivider . . . . .	205
14.6.8	GPT_GetClockDivider . . . . .	205

# Contents

Section Number	Title	Page Number
14.6.9	GPT_SetOscClockDivider . . . . .	206
14.6.10	GPT_GetOscClockDivider . . . . .	206
14.6.11	GPT_StartTimer . . . . .	206
14.6.12	GPT_StopTimer . . . . .	206
14.6.13	GPT_GetCurrentTimerCount . . . . .	207
14.6.14	GPT_SetInputOperationMode . . . . .	207
14.6.15	GPT_GetInputOperationMode . . . . .	207
14.6.16	GPT_GetInputCaptureValue . . . . .	208
14.6.17	GPT_SetOutputOperationMode . . . . .	209
14.6.18	GPT_GetOutputOperationMode . . . . .	209
14.6.19	GPT_SetOutputCompareValue . . . . .	209
14.6.20	GPT_GetOutputCompareValue . . . . .	210
14.6.21	GPT_ForceOutput . . . . .	210
14.6.22	GPT_EnableInterrupts . . . . .	210
14.6.23	GPT_DisableInterrupts . . . . .	211
14.6.24	GPT_GetEnabledInterrupts . . . . .	211
14.6.25	GPT_GetStatusFlags . . . . .	211
14.6.26	GPT_ClearStatusFlags . . . . .	212

## Chapter **GPC: General Power Controller Driver**

<b>15.1</b>	<b>Overview</b> . . . . .	<b>215</b>
<b>15.2</b>	<b>Macro Definition Documentation</b> . . . . .	<b>215</b>
15.2.1	FSL_GPC_DRIVER_VERSION . . . . .	215
<b>15.3</b>	<b>Function Documentation</b> . . . . .	<b>216</b>
15.3.1	GPC_AllowIRQs . . . . .	216
15.3.2	GPC_DisallowIRQs . . . . .	217
15.3.3	GPC_EnableIRQ . . . . .	217
15.3.4	GPC_DisableIRQ . . . . .	217
15.3.5	GPC_GetIRQStatusFlag . . . . .	217
15.3.6	GPC_RequestL2CachePowerDown . . . . .	218
15.3.7	GPC_RequestVADCPowerDown . . . . .	218
15.3.8	GPC_GetVADCPowerDownFlag . . . . .	218
15.3.9	GPC_HasDVFS0ChangeRequest . . . . .	218
15.3.10	GPC_RequestDisplayPowerOn . . . . .	219
15.3.11	GPC_RequestMEGAPowerOn . . . . .	219

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

<b>16.1</b>	<b>Overview</b> . . . . .	<b>221</b>
<b>16.2</b>	<b>GPIO Driver</b> . . . . .	<b>222</b>
16.2.1	Overview . . . . .	222

# Contents

Section Number	Title	Page Number
16.2.2	Typical use case . . . . .	222
16.2.3	Data Structure Documentation . . . . .	223
16.2.4	Macro Definition Documentation . . . . .	224
16.2.5	Enumeration Type Documentation . . . . .	224
16.2.6	Function Documentation . . . . .	224
<b>Chapter I2C: Inter-Integrated Circuit Driver</b>		
17.1	<b>Overview</b> . . . . .	229
17.2	<b>I2C Driver</b> . . . . .	230
17.2.1	Overview . . . . .	230
17.2.2	Typical use case . . . . .	230
17.2.3	Data Structure Documentation . . . . .	236
17.2.4	Macro Definition Documentation . . . . .	240
17.2.5	Typedef Documentation . . . . .	240
17.2.6	Enumeration Type Documentation . . . . .	240
17.2.7	Function Documentation . . . . .	242
17.3	<b>I2C FreeRTOS Driver</b> . . . . .	255
17.3.1	Overview . . . . .	255
17.3.2	Function Documentation . . . . .	255
<b>Chapter PWM: Pulse Width Modulation Driver</b>		
18.1	<b>Overview</b> . . . . .	257
18.2	<b>PWM Driver</b> . . . . .	258
18.2.1	Overview . . . . .	258
18.2.2	Typical use case . . . . .	258
18.2.3	Enumeration Type Documentation . . . . .	262
18.2.4	Function Documentation . . . . .	264
<b>Chapter UART: Universal Asynchronous Receiver/Transmitter Driver</b>		
19.1	<b>Overview</b> . . . . .	271
19.2	<b>UART Driver</b> . . . . .	272
19.2.1	Overview . . . . .	272
19.2.2	Typical use case . . . . .	272
19.2.3	Data Structure Documentation . . . . .	282
19.2.4	Macro Definition Documentation . . . . .	284
19.2.5	Typedef Documentation . . . . .	284
19.2.6	Enumeration Type Documentation . . . . .	284
19.2.7	Function Documentation . . . . .	286

# Contents

Section Number	Title	Page Number
19.2.8	Variable Documentation . . . . .	301
<b>19.3</b>	<b>UART FreeRTOS Driver . . . . .</b>	<b>302</b>
19.3.1	Overview . . . . .	302
19.3.2	Data Structure Documentation . . . . .	302
19.3.3	Function Documentation . . . . .	303
<b>Chapter MMDC: Multi Mode DDR Controller Driver</b>		
<b>20.1</b>	<b>Overview . . . . .</b>	<b>307</b>
<b>20.2</b>	<b>Typical use case . . . . .</b>	<b>307</b>
<b>20.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>314</b>
20.3.1	struct mmdc_readDQS_calibration_config_t . . . . .	314
20.3.2	struct mmdc_writeLeveling_calibration_config_t . . . . .	315
20.3.3	struct mmdc_read_calibration_config_t . . . . .	316
20.3.4	struct mmdc_fine_tuning_config_t . . . . .	316
20.3.5	struct mmdc_odt_config_t . . . . .	317
20.3.6	struct mmdc_power_config_t . . . . .	318
20.3.7	struct mmdc_zq_config_t . . . . .	319
20.3.8	struct mmdc_cmd_config_t . . . . .	319
20.3.9	struct mmdc_device_timing_t . . . . .	320
20.3.10	struct mmdc_auto_refresh_t . . . . .	322
20.3.11	struct mmdc_exaccess_config_t . . . . .	322
20.3.12	struct mmdc_profiling_config_t . . . . .	323
20.3.13	struct mmdc_performance_config_t . . . . .	323
20.3.14	struct mmdc_device_config_t . . . . .	324
20.3.15	struct mmdc_config_t . . . . .	324
<b>20.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>325</b>
20.4.1	MMDC_READ_DQS_FINE_TUNING_MASK . . . . .	325
20.4.2	MMDC_WRITE_DQS_FINE_TUNING_MASK . . . . .	325
20.4.3	MMDC_PRE_DEFINE_VALUE_DEFAULT . . . . .	325
20.4.4	MMDC_MEASUREUNIT_ERR_FREQ . . . . .	325
<b>20.5</b>	<b>Typedef Documentation . . . . .</b>	<b>326</b>
20.5.1	MMDC_SwitchFrequency . . . . .	326
<b>20.6</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>326</b>
20.6.1	_mmdc_status . . . . .	326
20.6.2	mmdc_device_type_t . . . . .	326
20.6.3	mmdc_device_bank_num_t . . . . .	326
20.6.4	mmdc_row_addr_width_t . . . . .	326
20.6.5	mmdc_col_addr_width_t . . . . .	327
20.6.6	mmdc_burst_len_t . . . . .	327

# Contents

Section Number	Title	Page Number
20.6.7	<code>mmdc_cmd_type_t</code>	327
20.6.8	<code>mmdc_zq_calmode_t</code>	327
20.6.9	<code>mmdc_zq_calfreq_t</code>	328
20.6.10	<code>mmdc_refresh_sel_t</code>	328
20.6.11	<code>mmdc_profiling_action_t</code>	328
20.6.12	<code>mmdc_calibration_type_t</code>	328
20.6.13	<code>mmdc_calibaration_waitcycles_t</code>	328
20.6.14	<code>mmdc_fine_tuning_dutycycle_t</code>	329
20.6.15	<code>mmdc_termination_config_t</code>	329
20.6.16	<code>_mmdc_lpddr2_derate</code>	329
20.6.17	<code>_mmdc_exaccess_type</code>	329
<b>20.7</b>	<b>Function Documentation</b>	<b>330</b>
20.7.1	<code>MMDC_GetDefaultConfig</code>	330
20.7.2	<code>MMDC_Init</code>	331
20.7.3	<code>MMDC_Deinit</code>	331
20.7.4	<code>MMDC_HandleCommand</code>	332
20.7.5	<code>MMDC_GetReadData</code>	332
20.7.6	<code>MMDC_EnhancePerformance</code>	332
20.7.7	<code>MMDC_EnableAutoRefresh</code>	332
20.7.8	<code>MMDC_DisableAutoRefresh</code>	333
20.7.9	<code>MMDC_EnablePowerSaving</code>	333
20.7.10	<code>MMDC_DisablePowerSaving</code>	333
20.7.11	<code>MMDC_Profiling</code>	333
20.7.12	<code>MMDC_LPDDR2UpdateDerate</code>	334
20.7.13	<code>MMDC_MonitorLPDDR2OperationTemp</code>	334
20.7.14	<code>MMDC_ReadDQSGatingCalibration</code>	334
20.7.15	<code>MMDC_WriteLevelingCalibration</code>	335
20.7.16	<code>MMDC_WriteCalibration</code>	335
20.7.17	<code>MMDC_ReadCalibration</code>	336
20.7.18	<code>MMDC_DoFineTuning</code>	337
20.7.19	<code>MMDC_SetTiming</code>	337
20.7.20	<code>MMDC_DeviceInit</code>	337
20.7.21	<code>MMDC_EnterConfigurationMode</code>	338
20.7.22	<code>MMDC_DoZQCalibration</code>	339
20.7.23	<code>MMDC_EnableLowPowerMode</code>	339
20.7.24	<code>MMDC_EnableDVFSMode</code>	339
20.7.25	<code>MMDC_Reset</code>	340
20.7.26	<code>MMDC_SwitchDeviceFrequency</code>	340
20.7.27	<code>MMDC_EnableSBS</code>	340
20.7.28	<code>MMDC_TriggerSBS</code>	341
20.7.29	<code>MMDC_GetAXIAddrBySBS</code>	341
20.7.30	<code>MMDC_GetAXIAtributeBySBS</code>	341
20.7.31	<code>MMDC_EnableProfiling</code>	341
20.7.32	<code>MMDC_ResumeProfiling</code>	342

# Contents

Section Number	Title	Page Number
20.7.33	MMDC_ResetProfiling . . . . .	343
20.7.34	MMDC_ExclusiveAccess . . . . .	343

## Chapter PMU: Power Management Unit

<b>21.1</b>	<b>Overview . . . . .</b>	<b>345</b>
<b>21.2</b>	<b>Macro Definition Documentation . . . . .</b>	<b>347</b>
21.2.1	FSL_PMU_DRIVER_VERSION . . . . .	347
<b>21.3</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>348</b>
21.3.1	_pmu_status_flags . . . . .	348
21.3.2	pmu_1p1_weak_reference_source_t . . . . .	348
21.3.3	pmu_3p0_vbus_voltage_source_t . . . . .	348
21.3.4	pmu_core_reg_voltage_ramp_rate_t . . . . .	348
21.3.5	_pmu_power_gate . . . . .	349
21.3.6	pmu_power_bandgap_t . . . . .	349
<b>21.4</b>	<b>Function Documentation . . . . .</b>	<b>349</b>
21.4.1	PMU_1P1SetWeakReferenceSource . . . . .	349
21.4.2	PMU_1P1EnableWeakRegulator . . . . .	349
21.4.3	PMU_1P1SetRegulatorOutputVoltage . . . . .	350
21.4.4	PMU_1P1SetBrownoutOffsetVoltage . . . . .	350
21.4.5	PMU_1P1EnablePullDown . . . . .	350
21.4.6	PMU_1P1EnableCurrentLimit . . . . .	351
21.4.7	PMU_1P1EnableBrownout . . . . .	351
21.4.8	PMU_1P1EnableOutput . . . . .	351
21.4.9	PMU_3P0SetRegulatorOutputVoltage . . . . .	351
21.4.10	PMU_3P0SetVBusVoltageSource . . . . .	352
21.4.11	PMU_3P0SetBrownoutOffsetVoltage . . . . .	352
21.4.12	PMU_3P0EnableCurrentLimit . . . . .	352
21.4.13	PMU_3P0EnableBrownout . . . . .	353
21.4.14	PMU_3P0EnableOutput . . . . .	353
21.4.15	PMU_2P5EnableWeakRegulator . . . . .	353
21.4.16	PMU_2P5SetRegulatorOutputVoltage . . . . .	353
21.4.17	PMU_2P5SetBrownoutOffsetVoltage . . . . .	354
21.4.18	PMU_2P1EnablePullDown . . . . .	354
21.4.19	PMU_2P5EnableCurrentLimit . . . . .	354
21.4.20	PMU_2P5nableBrownout . . . . .	355
21.4.21	PMU_2P5EnableOutput . . . . .	355
21.4.22	PMU_CoreEnableIncreaseGateDrive . . . . .	355
21.4.23	PMU_CoreSetRegulatorVoltageRampRate . . . . .	355
21.4.24	PMU_CoreSetSOCDomainVoltage . . . . .	356
21.4.25	PMU_CoreSetARMCoreDomainVoltage . . . . .	356
21.4.26	PMU_GatePower . . . . .	357

# Contents

Section Number	Title	Page Number
21.4.27	PMU_UngatePower . . . . .	357
21.4.28	PMU_EnableLowPowerBandgap . . . . .	357
<b>Chapter PXP: Pixel Pipeline</b>		
<b>22.1</b>	<b>Overview . . . . .</b>	<b>359</b>
<b>22.2</b>	<b>Typical use case . . . . .</b>	<b>359</b>
22.2.1	PXP normal operation . . . . .	359
22.2.2	PXP operation queue . . . . .	360
<b>22.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>367</b>
22.3.1	struct ppx_output_buffer_config_t . . . . .	367
22.3.2	struct ppx_ps_buffer_config_t . . . . .	368
22.3.3	struct ppx_as_buffer_config_t . . . . .	369
22.3.4	struct ppx_as_blend_config_t . . . . .	369
22.3.5	struct ppx_csc2_config_t . . . . .	370
22.3.6	struct ppx_lut_config_t . . . . .	371
22.3.7	struct ppx_dither_final_lut_data_t . . . . .	372
22.3.8	struct ppx_dither_config_t . . . . .	372
<b>22.4</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>374</b>
22.4.1	_ppx_interrupt_enable . . . . .	374
22.4.2	_ppx_flags . . . . .	374
22.4.3	ppx_flip_mode_t . . . . .	374
22.4.4	ppx_rotate_position_t . . . . .	375
22.4.5	ppx_rotate_degree_t . . . . .	375
22.4.6	ppx_interlaced_output_mode_t . . . . .	375
22.4.7	ppx_output_pixel_format_t . . . . .	375
22.4.8	ppx_ps_pixel_format_t . . . . .	376
22.4.9	ppx_as_pixel_format_t . . . . .	376
22.4.10	ppx_alpha_mode_t . . . . .	376
22.4.11	ppx_rop_mode_t . . . . .	377
22.4.12	ppx_block_size_t . . . . .	377
22.4.13	ppx_csc1_mode_t . . . . .	377
22.4.14	ppx_csc2_mode_t . . . . .	378
22.4.15	ppx_lut_lookup_mode_t . . . . .	378
22.4.16	ppx_lut_out_mode_t . . . . .	378
22.4.17	ppx_lut_8k_bank_t . . . . .	378
22.4.18	ppx_ram_t . . . . .	378
22.4.19	_ppx_dither_mode . . . . .	379
22.4.20	_ppx_dither_lut_mode . . . . .	379
22.4.21	_ppx_dither_matrix_size . . . . .	379
<b>22.5</b>	<b>Function Documentation . . . . .</b>	<b>379</b>

# Contents

Section Number	Title	Page Number
22.5.1	PXP_Init . . . . .	379
22.5.2	PXP_Deinit . . . . .	380
22.5.3	PXP_Reset . . . . .	380
22.5.4	PXP_Start . . . . .	380
22.5.5	PXP_EnableLcdHandShake . . . . .	380
22.5.6	PXP_EnableContinousRun . . . . .	381
22.5.7	PXP_SetProcessBlockSize . . . . .	381
22.5.8	PXP_GetStatusFlags . . . . .	381
22.5.9	PXP_ClearStatusFlags . . . . .	382
22.5.10	PXP_GetAxiErrorId . . . . .	382
22.5.11	PXP_EnableInterrupts . . . . .	382
22.5.12	PXP_DisableInterrupts . . . . .	384
22.5.13	PXP_SetAlphaSurfaceBufferConfig . . . . .	384
22.5.14	PXP_SetAlphaSurfaceBlendConfig . . . . .	384
22.5.15	PXP_SetAlphaSurfaceOverlayColorKey . . . . .	385
22.5.16	PXP_EnableAlphaSurfaceOverlayColorKey . . . . .	385
22.5.17	PXP_SetAlphaSurfacePosition . . . . .	385
22.5.18	PXP_SetProcessSurfaceBackGroundColor . . . . .	386
22.5.19	PXP_SetProcessSurfaceBufferConfig . . . . .	386
22.5.20	PXP_SetProcessSurfaceScaler . . . . .	386
22.5.21	PXP_SetProcessSurfacePosition . . . . .	387
22.5.22	PXP_SetProcessSurfaceColorKey . . . . .	387
22.5.23	PXP_SetOutputBufferConfig . . . . .	387
22.5.24	PXP_SetOverwrittenAlphaValue . . . . .	388
22.5.25	PXP_EnableOverWrittenAlpha . . . . .	388
22.5.26	PXP_SetRotateConfig . . . . .	388
22.5.27	PXP_SetNextCommand . . . . .	389
22.5.28	PXP_IsNextCommandPending . . . . .	390
22.5.29	PXP_CancelNextCommand . . . . .	390
22.5.30	PXP_SetCsc2Config . . . . .	390
22.5.31	PXP_EnableCsc2 . . . . .	390
22.5.32	PXP_SetCsc1Mode . . . . .	391
22.5.33	PXP_EnableCsc1 . . . . .	391
22.5.34	PXP_SetLutConfig . . . . .	391
22.5.35	PXP_LoadLutTable . . . . .	392
22.5.36	PXP_EnableLut . . . . .	392
22.5.37	PXP_Select8kLutBank . . . . .	392

## Chapter 23 QSPI: Quad Serial Peripheral Interface Driver

23.1	Overview . . . . .	395
23.2	Data Structure Documentation . . . . .	399
23.2.1	struct qspi_dqs_config_t . . . . .	399
23.2.2	struct qspi_flash_timing_t . . . . .	400

# Contents

Section Number	Title	Page Number
23.2.3	struct qspi_config_t . . . . .	400
23.2.4	struct qspi_flash_config_t . . . . .	401
23.2.5	struct qspi_transfer_t . . . . .	402
<b>23.3</b>	<b>Macro Definition Documentation</b> . . . . .	<b>402</b>
23.3.1	FSL_QSPI_DRIVER_VERSION . . . . .	402
<b>23.4</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>402</b>
23.4.1	_status_t . . . . .	402
23.4.2	qspi_read_area_t . . . . .	402
23.4.3	qspi_command_seq_t . . . . .	402
23.4.4	qspi_fifo_t . . . . .	403
23.4.5	qspi_endianness_t . . . . .	403
23.4.6	_qspi_error_flags . . . . .	403
23.4.7	_qspi_flags . . . . .	403
23.4.8	_qspi_interrupt_enable . . . . .	404
23.4.9	_qspi_dma_enable . . . . .	405
23.4.10	qspi_dqs_phrase_shift_t . . . . .	405
<b>23.5</b>	<b>Function Documentation</b> . . . . .	<b>405</b>
23.5.1	QSPI_Init . . . . .	405
23.5.2	QSPI_GetDefaultQspiConfig . . . . .	405
23.5.3	QSPI_Deinit . . . . .	405
23.5.4	QSPI_SetFlashConfig . . . . .	406
23.5.5	QSPI_SoftwareReset . . . . .	406
23.5.6	QSPI_Enable . . . . .	406
23.5.7	QSPI_GetStatusFlags . . . . .	406
23.5.8	QSPI_GetErrorStatusFlags . . . . .	407
23.5.9	QSPI_ClearErrorFlag . . . . .	407
23.5.10	QSPI_EnableInterrupts . . . . .	407
23.5.11	QSPI_DisableInterrupts . . . . .	408
23.5.12	QSPI_EnableDMA . . . . .	408
23.5.13	QSPI_GetTxDataRegisterAddress . . . . .	408
23.5.14	QSPI_GetRxDataRegisterAddress . . . . .	408
23.5.15	QSPI_SetIPCommandAddress . . . . .	409
23.5.16	QSPI_SetIPCommandSize . . . . .	409
23.5.17	QSPI_ExecuteIPCommand . . . . .	409
23.5.18	QSPI_ExecuteAHBCommand . . . . .	409
23.5.19	QSPI_EnableIPPParallelMode . . . . .	410
23.5.20	QSPI_EnableAHBParallelMode . . . . .	410
23.5.21	QSPI_UpdateLUT . . . . .	410
23.5.22	QSPI_ClearFifo . . . . .	410
23.5.23	QSPI_ClearCommandSequence . . . . .	411
23.5.24	QSPI_EnableDDRMode . . . . .	411
23.5.25	QSPI_SetReadDataArea . . . . .	411

# Contents

Section Number	Title	Page Number
23.5.26	QSPI_WriteBlocking . . . . .	411
23.5.27	QSPI_WriteData . . . . .	412
23.5.28	QSPI_ReadBlocking . . . . .	412
23.5.29	QSPI_ReadData . . . . .	412
23.5.30	QSPI_TransferSendBlocking . . . . .	413
23.5.31	QSPI_TransferReceiveBlocking . . . . .	413
<b>23.6</b>	<b>QSPI eDMA Driver . . . . .</b>	<b>414</b>
23.6.1	Overview . . . . .	414
23.6.2	Data Structure Documentation . . . . .	415
23.6.3	Function Documentation . . . . .	415
 <b>Chapter SAI: Serial Audio Interface</b>		
<b>24.1</b>	<b>Overview . . . . .</b>	<b>419</b>
<b>24.2</b>	<b>Typical use case . . . . .</b>	<b>419</b>
24.2.1	SAI Send/receive using an interrupt method . . . . .	419
24.2.2	SAI Send/receive using a DMA method . . . . .	420
<b>24.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>426</b>
24.3.1	struct sai_config_t . . . . .	426
24.3.2	struct sai_transfer_format_t . . . . .	426
24.3.3	struct sai_transfer_t . . . . .	427
24.3.4	struct _sai_handle . . . . .	427
<b>24.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>427</b>
24.4.1	SAI_XFER_QUEUE_SIZE . . . . .	427
<b>24.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>428</b>
24.5.1	_sai_status_t . . . . .	428
24.5.2	sai_protocol_t . . . . .	428
24.5.3	sai_master_slave_t . . . . .	428
24.5.4	sai_mono_stereo_t . . . . .	428
24.5.5	sai_sync_mode_t . . . . .	429
24.5.6	sai_mclk_source_t . . . . .	429
24.5.7	sai_bclk_source_t . . . . .	429
24.5.8	_sai_interrupt_enable_t . . . . .	429
24.5.9	_sai_dma_enable_t . . . . .	429
24.5.10	_sai_flags . . . . .	430
24.5.11	sai_reset_type_t . . . . .	430
24.5.12	sai_sample_rate_t . . . . .	430
24.5.13	sai_word_width_t . . . . .	430
<b>24.6</b>	<b>Function Documentation . . . . .</b>	<b>431</b>
24.6.1	SAI_TxInit . . . . .	431

# Contents

Section Number	Title	Page Number
24.6.2	SAI_RxInit . . . . .	431
24.6.3	SAI_TxGetDefaultConfig . . . . .	431
24.6.4	SAI_RxGetDefaultConfig . . . . .	432
24.6.5	SAI_Deinit . . . . .	432
24.6.6	SAI_TxReset . . . . .	432
24.6.7	SAI_RxReset . . . . .	432
24.6.8	SAI_TxEnable . . . . .	433
24.6.9	SAI_RxEnable . . . . .	433
24.6.10	SAI_TxGetStatusFlag . . . . .	433
24.6.11	SAI_TxClearStatusFlags . . . . .	433
24.6.12	SAI_RxGetStatusFlag . . . . .	434
24.6.13	SAI_RxClearStatusFlags . . . . .	434
24.6.14	SAI_TxSoftwareReset . . . . .	434
24.6.15	SAI_RxSoftwareReset . . . . .	435
24.6.16	SAI_TxSetChannelFIFOMask . . . . .	435
24.6.17	SAI_RxSetChannelFIFOMask . . . . .	435
24.6.18	SAI_TxEnableInterrupts . . . . .	435
24.6.19	SAI_RxEnableInterrupts . . . . .	436
24.6.20	SAI_TxDisableInterrupts . . . . .	437
24.6.21	SAI_RxDisableInterrupts . . . . .	437
24.6.22	SAI_TxEnableDMA . . . . .	438
24.6.23	SAI_RxEnableDMA . . . . .	438
24.6.24	SAI_TxGetDataRegisterAddress . . . . .	438
24.6.25	SAI_RxGetDataRegisterAddress . . . . .	439
24.6.26	SAI_TxSetFormat . . . . .	439
24.6.27	SAI_RxSetFormat . . . . .	440
24.6.28	SAI_WriteBlocking . . . . .	440
24.6.29	SAI_WriteData . . . . .	440
24.6.30	SAI_ReadBlocking . . . . .	441
24.6.31	SAI_ReadData . . . . .	441
24.6.32	SAI_TransferTxCreateHandle . . . . .	441
24.6.33	SAI_TransferRxCreateHandle . . . . .	442
24.6.34	SAI_TransferTxSetFormat . . . . .	442
24.6.35	SAI_TransferRxSetFormat . . . . .	443
24.6.36	SAI_TransferSendNonBlocking . . . . .	443
24.6.37	SAI_TransferReceiveNonBlocking . . . . .	444
24.6.38	SAI_TransferGetSendCount . . . . .	444
24.6.39	SAI_TransferGetReceiveCount . . . . .	445
24.6.40	SAI_TransferAbortSend . . . . .	445
24.6.41	SAI_TransferAbortReceive . . . . .	446
24.6.42	SAI_TransferTerminateSend . . . . .	446
24.6.43	SAI_TransferTerminateReceive . . . . .	446
24.6.44	SAI_TransferTxHandleIRQ . . . . .	447
24.6.45	SAI_TransferRxHandleIRQ . . . . .	447

# Contents

Section Number	Title	Page Number
<b>24.7</b>	<b>SAI DMA Driver . . . . .</b>	<b>448</b>
24.7.1	Overview . . . . .	448
24.7.2	Data Structure Documentation . . . . .	449
24.7.3	Function Documentation . . . . .	449
<b>24.8</b>	<b>SAI eDMA Driver . . . . .</b>	<b>455</b>
24.8.1	Overview . . . . .	455
24.8.2	Data Structure Documentation . . . . .	456
24.8.3	Function Documentation . . . . .	457
<b>24.9</b>	<b>SAI SDMA Driver . . . . .</b>	<b>464</b>
24.9.1	Overview . . . . .	464
24.9.2	Data Structure Documentation . . . . .	465
24.9.3	Function Documentation . . . . .	466
 <b>Chapter SDMA: Smart Direct Memory Access (SDMA) Controller Driver</b>		
<b>25.1</b>	<b>Overview . . . . .</b>	<b>471</b>
<b>25.2</b>	<b>Typical use case . . . . .</b>	<b>471</b>
25.2.1	SDMA Operation . . . . .	471
<b>25.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>475</b>
25.3.1	struct sdma_config_t . . . . .	475
25.3.2	struct sdma_transfer_config_t . . . . .	475
25.3.3	struct sdma_buffer_descriptor_t . . . . .	476
25.3.4	struct sdma_channel_control_t . . . . .	476
25.3.5	struct sdma_context_data_t . . . . .	477
25.3.6	struct sdma_handle_t . . . . .	477
<b>25.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>478</b>
25.4.1	FSL_SDMA_DRIVER_VERSION . . . . .	478
<b>25.5</b>	<b>Typedef Documentation . . . . .</b>	<b>478</b>
25.5.1	sdma_callback . . . . .	478
<b>25.6</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>478</b>
25.6.1	sdma_transfer_size_t . . . . .	478
25.6.2	sdma_bd_status_t . . . . .	478
25.6.3	sdma_bd_command_t . . . . .	479
25.6.4	sdma_context_switch_mode_t . . . . .	479
25.6.5	sdma_clock_ratio_t . . . . .	479
25.6.6	sdma_transfer_type_t . . . . .	479
25.6.7	sdma_peripheral_t . . . . .	480
25.6.8	_sdma_transfer_status . . . . .	480

# Contents

Section Number	Title	Page Number
<b>25.7</b>	<b>Function Documentation</b>	<b>480</b>
25.7.1	SDMA_Init . . . . .	480
25.7.2	SDMA_Deinit . . . . .	480
25.7.3	SDMA_GetDefaultConfig . . . . .	481
25.7.4	SDMA_ResetModule . . . . .	481
25.7.5	SDMA_EnableChannelErrorInterrupts . . . . .	481
25.7.6	SDMA_DisableChannelErrorInterrupts . . . . .	481
25.7.7	SDMA_ConfigBufferDescriptor . . . . .	482
25.7.8	SDMA_SetChannelPriority . . . . .	482
25.7.9	SDMA_SetSourceChannel . . . . .	483
25.7.10	SDMA_StartChannelSoftware . . . . .	484
25.7.11	SDMA_StartChannelEvents . . . . .	484
25.7.12	SDMA_StopChannel . . . . .	484
25.7.13	SDMA_SetContextSwitchMode . . . . .	485
25.7.14	SDMA_GetChannelInterruptStatus . . . . .	486
25.7.15	SDMA_ClearChannelInterruptStatus . . . . .	486
25.7.16	SDMA_GetChannelStopStatus . . . . .	486
25.7.17	SDMA_ClearChannelStopStatus . . . . .	486
25.7.18	SDMA_GetChannelPendStatus . . . . .	487
25.7.19	SDMA_ClearChannelPendStatus . . . . .	487
25.7.20	SDMA_GetErrorStatus . . . . .	487
25.7.21	SDMA_GetRequestSourceStatus . . . . .	488
25.7.22	SDMA_CreateHandle . . . . .	489
25.7.23	SDMA_InstallBDMemory . . . . .	489
25.7.24	SDMA_SetCallback . . . . .	490
25.7.25	SDMA_PreparesTransfer . . . . .	491
25.7.26	SDMA_SubmitTransfer . . . . .	491
25.7.27	SDMA_StartTransfer . . . . .	492
25.7.28	SDMA_StopTransfer . . . . .	492
25.7.29	SDMA_AbortTransfer . . . . .	492
25.7.30	SDMA_HandleIRQ . . . . .	492

## Chapter SNVS\_HP: Secure Non-Volatile Storage

<b>26.1</b>	<b>Overview . . . . .</b>	<b>495</b>
<b>26.2</b>	<b>SNVS_HP Driver Initialization and Configuration . . . . .</b>	<b>495</b>
<b>26.3</b>	<b>Set &amp; Get Datetime . . . . .</b>	<b>495</b>
<b>26.4</b>	<b>Set &amp; Get Alarm . . . . .</b>	<b>496</b>
<b>26.5</b>	<b>Start &amp; Stop timer . . . . .</b>	<b>496</b>
<b>26.6</b>	<b>Status . . . . .</b>	<b>496</b>

# Contents

Section Number	Title	Page Number
<b>26.7</b>	<b>Interrupt</b>	<b>496</b>
<b>26.8</b>	<b>Typical use case</b>	<b>496</b>
26.8.1	SNVS_HP RTC example	496
<b>26.9</b>	<b>Data Structure Documentation</b>	<b>499</b>
26.9.1	struct snvs_hp_RTC_datetime_t	499
26.9.2	struct snvs_hp_RTC_config_t	500
<b>26.10</b>	<b>Enumeration Type Documentation</b>	<b>500</b>
26.10.1	snvs_hp_interrupt_enable_t	500
26.10.2	snvs_hp_status_flags_t	500
<b>26.11</b>	<b>Function Documentation</b>	<b>500</b>
26.11.1	SNVS_HP_RTC_Init	500
26.11.2	SNVS_HP_RTC_Deinit	501
26.11.3	SNVS_HP_RTC_GetDefaultConfig	501
26.11.4	SNVS_HP_RTC_SetDatetime	501
26.11.5	SNVS_HP_RTC_GetDatetime	502
26.11.6	SNVS_HP_RTC_SetAlarm	502
26.11.7	SNVS_HP_RTC_GetAlarm	502
26.11.8	SNVS_HP_RTC_EnableInterrupts	503
26.11.9	SNVS_HP_RTC_DisableInterrupts	503
26.11.10	SNVS_HP_RTC_GetEnabledInterrupts	503
26.11.11	SNVS_HP_RTC_GetStatusFlags	503
26.11.12	SNVS_HP_RTC_ClearStatusFlags	504
26.11.13	SNVS_HP_RTC_StartTimer	504
26.11.14	SNVS_HP_RTC_StopTimer	504
<b>26.12</b>	<b>Variable Documentation</b>	<b>505</b>
26.12.1	year	505
26.12.2	month	505
26.12.3	day	505
26.12.4	hour	505
26.12.5	minute	505
26.12.6	second	505
 <b>Chapter SRC: System Reset Controller Driver</b>		
<b>27.1</b>	<b>Overview</b>	<b>507</b>
<b>27.2</b>	<b>Macro Definition Documentation</b>	<b>509</b>
27.2.1	FSL_SRC_DRIVER_VERSION	509
<b>27.3</b>	<b>Enumeration Type Documentation</b>	<b>509</b>
27.3.1	_src_reset_status_flags	509

# Contents

Section Number	Title	Page Number
27.3.2	<a href="#">_src_status_flags</a>	509
27.3.3	<a href="#">src_mix_reset_stretch_cycles_t</a>	509
27.3.4	<a href="#">src_wdog3_reset_option_t</a>	510
27.3.5	<a href="#">src_warm_reset_bypass_count_t</a>	510
<b>27.4</b>	<b>Function Documentation</b>	<b>510</b>
27.4.1	<a href="#">SRC_EnableWDOG3Reset</a>	510
27.4.2	<a href="#">SRC_SetMixResetStretchCycles</a>	510
27.4.3	<a href="#">SRC_EnableCoreDebugResetAfterPowerGate</a>	511
27.4.4	<a href="#">SRC_SetWdog3ResetOption</a>	511
27.4.5	<a href="#">SRC_DoSoftwareResetARMCoreDebug</a>	511
27.4.6	<a href="#">SRC_GetSoftwareResetARMCoreDebugDone</a>	511
27.4.7	<a href="#">SRC_DoSoftwareResetARMCore0</a>	512
27.4.8	<a href="#">SRC_GetSoftwareResetARMCore0Done</a>	512
27.4.9	<a href="#">SRC_AssertEIMReset</a>	512
27.4.10	<a href="#">SRC_EnableWDOGReset</a>	512
27.4.11	<a href="#">SRC_SetWarmResetBypassCount</a>	514
27.4.12	<a href="#">SRC_EnableWarmReset</a>	514
27.4.13	<a href="#">SRC_GetStatusFlags</a>	514
27.4.14	<a href="#">SRC_GetBootModeWord1</a>	515
27.4.15	<a href="#">SRC_GetBootModeWord2</a>	516
27.4.16	<a href="#">SRC_SetWarmBootIndication</a>	516
27.4.17	<a href="#">SRC_GetResetStatusFlags</a>	516
27.4.18	<a href="#">SRC_ClearResetStatusFlags</a>	517
27.4.19	<a href="#">SRC_SetGeneralPurposeRegister</a>	517
27.4.20	<a href="#">SRC_GetGeneralPurposeRegister</a>	517

## Chapter [TSC: Touch Screen Controller Driver](#)

<b>28.1</b>	<b>Overview</b>	<b>519</b>
<b>28.2</b>	<b>Typical use case</b>	<b>519</b>
28.2.1	<a href="#">4-wire Polling Configuration</a>	519
28.2.2	<a href="#">4-wire Interrupt Configuration</a>	520
<b>28.3</b>	<b>Data Structure Documentation</b>	<b>523</b>
28.3.1	<a href="#">struct tsc_config_t</a>	523
<b>28.4</b>	<b>Macro Definition Documentation</b>	<b>524</b>
28.4.1	<a href="#">FSL_TSC_DRIVER_VERSION</a>	524
<b>28.5</b>	<b>Enumeration Type Documentation</b>	<b>524</b>
28.5.1	<a href="#">tsc_detection_mode_t</a>	524
28.5.2	<a href="#">tsc_coordinate_value_selection_t</a>	524
28.5.3	<a href="#">_tsc_interrupt_signal_mask</a>	524

# Contents

Section Number	Title	Page Number
28.5.4	_tsc_interrupt_mask . . . . .	525
28.5.5	_tsc_interrupt_status_flag_mask . . . . .	525
28.5.6	_tsc_adc_status_flag_mask . . . . .	525
28.5.7	_tsc_status_flag_mask . . . . .	525
28.5.8	tsc_state_machine_t . . . . .	526
28.5.9	tsc_glitch_threshold_t . . . . .	526
28.5.10	tsc_trigger_signal_t . . . . .	526
28.5.11	tsc_port_source_t . . . . .	527
28.5.12	tsc_port_mode_t . . . . .	527
<b>28.6</b>	<b>Function Documentation</b> . . . . .	<b>527</b>
28.6.1	TSC_Init . . . . .	527
28.6.2	TSC_Deinit . . . . .	527
<b>28.7</b>	<b>Variable Documentation</b> . . . . .	<b>528</b>
28.7.1	enableAutoMeasure . . . . .	528
28.7.2	measureDelayTime . . . . .	528
28.7.3	prechargeTime . . . . .	528
28.7.4	detectionMode . . . . .	528

## Chapter **USDH C: ultra Secured Digital Host Controller Driver**

<b>29.1</b>	<b>Overview</b> . . . . .	<b>529</b>
<b>29.2</b>	<b>Typical use case</b> . . . . .	<b>529</b>
29.2.1	USDH C Operation . . . . .	529
<b>29.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>539</b>
29.3.1	struct usdh c_adma2_descriptor_t . . . . .	539
29.3.2	struct usdh c_capability_t . . . . .	539
29.3.3	struct usdh c_boot_config_t . . . . .	539
29.3.4	struct usdh c_config_t . . . . .	540
29.3.5	struct usdh c_data_t . . . . .	541
29.3.6	struct usdh c_command_t . . . . .	541
29.3.7	struct usdh c_adma_config_t . . . . .	542
29.3.8	struct usdh c_transfer_t . . . . .	542
29.3.9	struct usdh c_transfer_callback_t . . . . .	542
29.3.10	struct _usdh c_handle . . . . .	542
29.3.11	struct usdh c_host_t . . . . .	543
<b>29.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>543</b>
29.4.1	FSL_USDH C_DRIVER_VERSION . . . . .	543
<b>29.5</b>	<b>Typedef Documentation</b> . . . . .	<b>544</b>
29.5.1	usdh c_adma1_descriptor_t . . . . .	544
29.5.2	usdh c_transfer_function_t . . . . .	544

# Contents

Section Number	Title	Page Number
<b>29.6</b>	<b>Enumeration Type Documentation</b>	<b>544</b>
29.6.1	_usdhc_status . . . . .	544
29.6.2	_usdhc_capability_flag . . . . .	544
29.6.3	_usdhc_wakeup_event . . . . .	544
29.6.4	_usdhc_reset . . . . .	545
29.6.5	_usdhc_transfer_flag . . . . .	545
29.6.6	_usdhc_present_status_flag . . . . .	545
29.6.7	_usdhc_interrupt_status_flag . . . . .	546
29.6.8	_usdhc_auto_command12_error_status_flag . . . . .	547
29.6.9	_usdhc_standard_tuning . . . . .	547
29.6.10	_usdhc_adma_error_status_flag . . . . .	547
29.6.11	usdhc_adma_error_state_t . . . . .	547
29.6.12	_usdhc_force_event . . . . .	548
29.6.13	usdhc_data_bus_width_t . . . . .	548
29.6.14	usdhc_endian_mode_t . . . . .	548
29.6.15	usdhc_dma_mode_t . . . . .	549
29.6.16	_usdhc_sdio_control_flag . . . . .	549
29.6.17	usdhc_boot_mode_t . . . . .	549
29.6.18	usdhc_card_command_type_t . . . . .	549
29.6.19	usdhc_card_response_type_t . . . . .	549
29.6.20	_usdhc_adma1_descriptor_flag . . . . .	550
29.6.21	_usdhc_adma2_descriptor_flag . . . . .	550
29.6.22	usdhc_burst_len_t . . . . .	550
<b>29.7</b>	<b>Function Documentation</b>	<b>551</b>
29.7.1	USDHC_Init . . . . .	551
29.7.2	USDHC_Deinit . . . . .	551
29.7.3	USDHC_Reset . . . . .	551
29.7.4	USDHC_SetAdmaTableConfig . . . . .	552
29.7.5	USDHC_EnableInterruptStatus . . . . .	552
29.7.6	USDHC_DisableInterruptStatus . . . . .	552
29.7.7	USDHC_EnableInterruptSignal . . . . .	553
29.7.8	USDHC_DisableInterruptSignal . . . . .	553
29.7.9	USDHC_GetInterruptStatusFlags . . . . .	553
29.7.10	USDHC_ClearInterruptStatusFlags . . . . .	553
29.7.11	USDHC_GetAutoCommand12ErrorStatusFlags . . . . .	554
29.7.12	USDHC_GetAdmaErrorStatusFlags . . . . .	554
29.7.13	USDHC_GetPresentStatusFlags . . . . .	554
29.7.14	USDHC_GetCapability . . . . .	555
29.7.15	USDHC_ForceClockOn . . . . .	556
29.7.16	USDHC_SetSdClock . . . . .	556
29.7.17	USDHC_SetCardActive . . . . .	556
29.7.18	USDHC_AssertHardwareReset . . . . .	557
29.7.19	USDHC_SetDataBusWidth . . . . .	557
29.7.20	USDHC_WriteData . . . . .	557

# Contents

Section Number	Title	Page Number
29.7.21	USDHC_ReadData . . . . .	557
29.7.22	USDHC_SendCommand . . . . .	558
29.7.23	USDHC_EnableWakeupEvent . . . . .	558
29.7.24	USDHC_CardDetectByData3 . . . . .	558
29.7.25	USDHC_DetectCardInsert . . . . .	558
29.7.26	USDHC_EnableSdioControl . . . . .	559
29.7.27	USDHC_SetContinueRequest . . . . .	559
29.7.28	USDHC_SetMmcBootConfig . . . . .	559
29.7.29	USDHC_SetForceEvent . . . . .	560
29.7.30	UDSHC_SelectVoltage . . . . .	561
29.7.31	USDHC_RequestTuningForSDR50 . . . . .	561
29.7.32	USDHC_RequestReTuning . . . . .	561
29.7.33	USDHC_EnableAutoTuning . . . . .	561
29.7.34	USDHC_SetRetuningTimer . . . . .	562
29.7.35	USDHC_EnableDDRMode . . . . .	562
29.7.36	USDHC_EnableAutoTuningForCmdAndData . . . . .	562
29.7.37	USDHC_EnableManualTuning . . . . .	562
29.7.38	USDHC_AdjustDelayForManualTuning . . . . .	563
29.7.39	USDHC_EnableStandardTuning . . . . .	563
29.7.40	USDHC_GetExecuteStdTuningStatus . . . . .	563
29.7.41	USDHC_CheckStdTuningResult . . . . .	564
29.7.42	USDHC_CheckTuningError . . . . .	564
29.7.43	USDHC_TransferBlocking . . . . .	564
29.7.44	USDHC_TransferCreateHandle . . . . .	565
29.7.45	USDHC_TransferNonBlocking . . . . .	565
29.7.46	USDHC_TransferHandleIRQ . . . . .	566

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

<b>30.1</b>	<b>Overview</b> . . . . .	<b>567</b>
<b>30.2</b>	<b>Typical use case</b> . . . . .	<b>567</b>
<b>30.3</b>	<b>Data Structure Documentation</b> . . . . .	<b>569</b>
30.3.1	struct wdog_work_mode_t . . . . .	569
30.3.2	struct wdog_config_t . . . . .	569
30.3.3	struct wdog_test_config_t . . . . .	570
<b>30.4</b>	<b>Macro Definition Documentation</b> . . . . .	<b>570</b>
30.4.1	FSL_WDOG_DRIVER_VERSION . . . . .	570
<b>30.5</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>570</b>
30.5.1	wdog_clock_source_t . . . . .	570
30.5.2	wdog_clock_prescaler_t . . . . .	570
30.5.3	wdog_test_mode_t . . . . .	571

# Contents

Section Number	Title	Page Number
30.5.4	wdog_tested_byte_t . . . . .	571
30.5.5	_wdog_interrupt_enable_t . . . . .	571
30.5.6	_wdog_status_flags_t . . . . .	571
<b>30.6</b>	<b>Function Documentation</b> . . . . .	<b>571</b>
30.6.1	WDOG_GetDefaultConfig . . . . .	571
30.6.2	WDOG_Init . . . . .	572
30.6.3	WDOG_Deinit . . . . .	572
30.6.4	WDOG_SetTestModeConfig . . . . .	573
30.6.5	WDOG_Enable . . . . .	573
30.6.6	WDOG_Disable . . . . .	573
30.6.7	WDOG_EnableInterrupts . . . . .	574
30.6.8	WDOG_DisableInterrupts . . . . .	574
30.6.9	WDOG_GetStatusFlags . . . . .	574
30.6.10	WDOG_ClearStatusFlags . . . . .	575
30.6.11	WDOG_SetTimeoutValue . . . . .	575
30.6.12	WDOG_SetWindowValue . . . . .	576
30.6.13	WDOG_Unlock . . . . .	576
30.6.14	WDOG_Refresh . . . . .	576
30.6.15	WDOG_GetResetCount . . . . .	577
30.6.16	WDOG_ClearResetCount . . . . .	578

## Chapter Clock Driver

<b>31.1</b>	<b>Overview</b> . . . . .	<b>579</b>
<b>31.2</b>	<b>Get frequency</b> . . . . .	<b>579</b>
<b>31.3</b>	<b>External clock frequency</b> . . . . .	<b>579</b>
<b>31.4</b>	<b>Data Structure Documentation</b> . . . . .	<b>589</b>
31.4.1	struct clock_arm_pll_config_t . . . . .	589
31.4.2	struct clock_usb_pll_config_t . . . . .	589
31.4.3	struct clock_sys_pll_config_t . . . . .	590
31.4.4	struct clock_audio_pll_config_t . . . . .	590
31.4.5	struct clock_video_pll_config_t . . . . .	590
31.4.6	struct clock_enet_pll_config_t . . . . .	591
<b>31.5</b>	<b>Macro Definition Documentation</b> . . . . .	<b>592</b>
31.5.1	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL . . . . .	592
31.5.2	FSL_CLOCK_DRIVER_VERSION . . . . .	592
31.5.3	ADC_CLOCKS . . . . .	592
31.5.4	ADC_5HC_CLOCKS . . . . .	592
31.5.5	ECSPI_CLOCKS . . . . .	592
31.5.6	ENET_CLOCKS . . . . .	593

# Contents

Section Number	Title	Page Number
31.5.7	EPIT_CLOCKS . . . . .	593
31.5.8	FLEXCAN_CLOCKS . . . . .	593
31.5.9	FLEXCAN_PERIPH_CLOCKS . . . . .	593
31.5.10	GPIO_CLOCKS . . . . .	593
31.5.11	GPT_CLOCKS . . . . .	594
31.5.12	GPT_PERIPH_CLOCKS . . . . .	594
31.5.13	I2C_CLOCKS . . . . .	594
31.5.14	PWM_CLOCKS . . . . .	594
31.5.15	QSPI_CLOCKS . . . . .	594
31.5.16	SAI_CLOCKS . . . . .	595
31.5.17	SDMA_CLOCKS . . . . .	595
31.5.18	SIM_CLOCKS . . . . .	595
31.5.19	TSC_CLOCKS . . . . .	595
31.5.20	UART_CLOCKS . . . . .	595
31.5.21	USDHC_CLOCKS . . . . .	596
31.5.22	WDOG_CLOCKS . . . . .	596
31.5.23	LCDIF_CLOCKS . . . . .	596
31.5.24	LCDIF_PERIPH_CLOCKS . . . . .	596
31.5.25	PXP_CLOCKS . . . . .	596
31.5.26	SNVS_HP_CLOCKS . . . . .	597
31.5.27	SNVS_LP_CLOCKS . . . . .	597
31.5.28	CSI_CLOCKS . . . . .	597
31.5.29	CSI_MCLK_CLOCKS . . . . .	597
31.5.30	FSL_CLOCK_MMDC_IPG_GATE_COUNT . . . . .	597
31.5.31	MMDC_ACLK_CLOCKS . . . . .	597
31.5.32	kCLOCK_CoreSysClk . . . . .	598
31.5.33	CLOCK_GetCoreSysClkFreq . . . . .	598
<b>31.6</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>598</b>
31.6.1	clock_name_t . . . . .	598
31.6.2	clock_ip_name_t . . . . .	598
31.6.3	clock_osc_t . . . . .	601
31.6.4	clock_gate_value_t . . . . .	601
31.6.5	clock_mode_t . . . . .	601
31.6.6	clock_mux_t . . . . .	601
31.6.7	clock_div_t . . . . .	602
31.6.8	clock_pll_t . . . . .	603
31.6.9	clock_pfd_t . . . . .	604
31.6.10	clock_usb_src_t . . . . .	604
31.6.11	clock_usb_phy_src_t . . . . .	604
<b>31.7</b>	<b>Function Documentation</b> . . . . .	<b>604</b>
31.7.1	CLOCK_SetMux . . . . .	604
31.7.2	CLOCK_GetMux . . . . .	605
31.7.3	CLOCK_SetDiv . . . . .	605

# Contents

Section Number	Title	Page Number
31.7.4	<a href="#">CLOCK_GetDiv</a>	605
31.7.5	<a href="#">CLOCK_ControlGate</a>	605
31.7.6	<a href="#">CLOCK_EnableClock</a>	606
31.7.7	<a href="#">CLOCK_DisableClock</a>	606
31.7.8	<a href="#">CLOCK_SetMode</a>	606
31.7.9	<a href="#">CLOCK_GetFreq</a>	606
31.7.10	<a href="#">CLOCK_InitExternalClk</a>	607
31.7.11	<a href="#">CLOCK_DeinitExternalClk</a>	607
31.7.12	<a href="#">CLOCK_SwitchOsc</a>	607
31.7.13	<a href="#">CLOCK_GetOscFreq</a>	607
31.7.14	<a href="#">CLOCK_GetRtcFreq</a>	608
31.7.15	<a href="#">CLOCK_SetXtalFreq</a>	608
31.7.16	<a href="#">CLOCK_SetRtcXtalFreq</a>	608
31.7.17	<a href="#">CLOCK_InitArmPll</a>	608
31.7.18	<a href="#">CLOCK_InitSysPll</a>	608
31.7.19	<a href="#">CLOCK_InitUsb1Pll</a>	609
31.7.20	<a href="#">CLOCK_InitUsb2Pll</a>	609
31.7.21	<a href="#">CLOCK_InitAudioPll</a>	609
31.7.22	<a href="#">CLOCK_InitVideoPll</a>	609
31.7.23	<a href="#">CLOCK_InitEnetPll</a>	610
31.7.24	<a href="#">CLOCK_DeinitEnetPll</a>	611
31.7.25	<a href="#">CLOCK_GetPllFreq</a>	611
31.7.26	<a href="#">CLOCK_InitSysPfd</a>	611
31.7.27	<a href="#">CLOCK_DeinitSysPfd</a>	611
31.7.28	<a href="#">CLOCK_InitUsb1Pfd</a>	612
31.7.29	<a href="#">CLOCK_DeinitUsb1Pfd</a>	612
31.7.30	<a href="#">CLOCK_GetSysPfdFreq</a>	612
31.7.31	<a href="#">CLOCK_GetUsb1PfdFreq</a>	612
31.7.32	<a href="#">CLOCK_EnableUsbhs0Clock</a>	613
31.7.33	<a href="#">CLOCK_EnableUsbhs0PhyPllClock</a>	613
31.7.34	<a href="#">CLOCK_DisableUsbhs0PhyPllClock</a>	614
31.7.35	<a href="#">CLOCK_EnableUsbhs1Clock</a>	614
31.7.36	<a href="#">CLOCK_EnableUsbhs1PhyPllClock</a>	614
31.7.37	<a href="#">CLOCK_DisableUsbhs1PhyPllClock</a>	614
<b>31.8</b>	<b><a href="#">Variable Documentation</a></b>	<b>615</b>
31.8.1	<a href="#">g_xtalFreq</a>	615
31.8.2	<a href="#">g_rtcXtalFreq</a>	615
<b>Chapter</b>	<b>Debug Console</b>	
<b>32.1</b>	<b><a href="#">Overview</a></b>	<b>617</b>
<b>32.2</b>	<b><a href="#">Function groups</a></b>	<b>617</b>
32.2.1	<a href="#">Initialization</a>	617

## Contents

Section Number	Title	Page Number
32.2.2	Advanced Feature . . . . .	618
<b>32.3</b>	<b>Typical use case . . . . .</b>	<b>621</b>
<b>32.4</b>	<b>Semihosting . . . . .</b>	<b>623</b>
32.4.1	Guide Semihosting for IAR . . . . .	623
32.4.2	Guide Semihosting for Keil µVision . . . . .	623
32.4.3	Guide Semihosting for KDS . . . . .	625
32.4.4	Guide Semihosting for ATL . . . . .	625
32.4.5	Guide Semihosting for ARMGCC . . . . .	626

Confidential Proprietary

# 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, USB stack, and integrated RTOS support for FreeRTOS, µC/OS-II, and µC/OS-III. In addition to the base enablement, the KSDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support provided by MCUXpresso SDK. The KEx Web UI is available to provide access to all MCUXpresso SDK packages. See the *MCUXpresso Software Development Kit (MCUXpresso SDK) Release Notes* (document SDK200RN) and 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 KSDK peripheral drivers and leverage native RTOS services to better comply to the RTOS cases.
- Real time operation systems (RTOS) including FreeRTOS OS, µC/OS-II, and µC/OS-III.
- Stacks and middleware in source or object formats including:
  - A USB device, host, and OTG stack with comprehensive USB class support.
  - CMSIS-DSP, a suite of common signal processing functions.
  - 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
- Keil MDK
- MCUXpresso IDE

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 [kex.-nxp.com/apidoc](#).

<b>Deliverable</b>	<b>Location</b>
Examples	<install_dir>/examples/
Demo Applications	<install_dir>/examples/<board_name>/demo_apps/
Driver Examples	<install_dir>/examples/<board_name>/driver_examples/
Documentation	<install_dir>/doc/
USB Documentation	<install_dir>/doc/usb/
Middleware	<install_dir>/middleware/
USB Stack	<install_dir>/middleware/usb_<version>
Drivers	<install_dir>/<device_name>/drivers/
CMSIS Standard ARM Cortex-M Headers, math and DSP Libraries	<install_dir>/<device_name>/CMSIS/
Device Startup and Linker	<install_dir>/<device_name>/<toolchain>/
SDK Utilities	<install_dir>/<device_name>/utilities/
RTOS Kernels	<install_dir>/rtos/

Table 2: MCUXpresso SDK Folder Structure

## Chapter 2

### Driver errors status

- `kStatus_CSI_NoEmptyBuffer` = 2900
- `kStatus_CSI_NoFullBuffer` = 2901
- `kStatus_CSI_QueueFull` = 2902
- `kStatus_ENET_RxFrameError` = 4000
- `kStatus_ENET_RxFrameFail` = 4001
- `kStatus_ENET_RxFrameEmpty` = 4002
- `kStatus_ENET_TxFrameBusy` = 4003
- `kStatus_ENET_TxFrameFail` = 4004
- `#kStatus_ENET_PtpTsRingFull` = 4005
- `#kStatus_ENET_PtpTsRingEmpty` = 4006
- `kStatus_QSPI_Idle` = 4500
- `kStatus_QSPI_Busy` = 4501
- `kStatus_QSPI_Error` = 4502
- `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_SDMA_ERROR` = 7300
- `kStatus_SDMA_Busy` = 7301



# 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

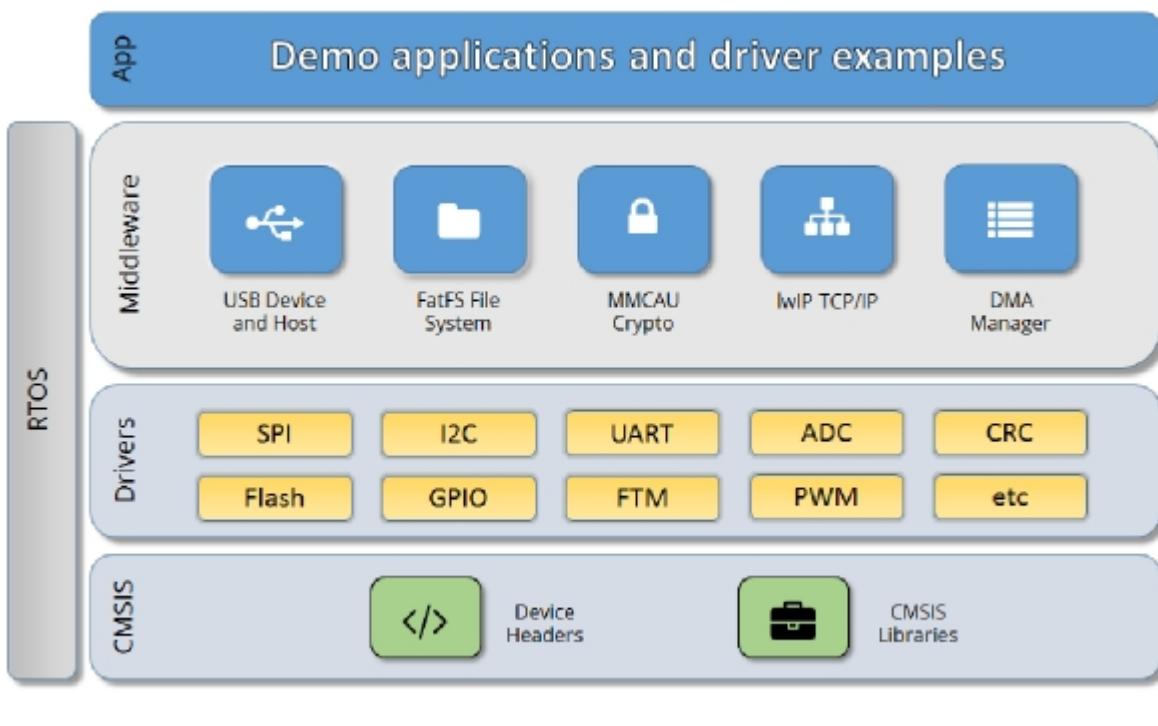


Figure 1: MCUXpresso SDK Block Diagram

### 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 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 file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0\_DriverIRQHandler) jumps to itself (BX). The MCUXpresso SDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0\_DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

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

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

## Feature Header Files

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

## Application

See the *Getting Started with MCUXpresso SDK* document (SDK20GSUG).



## **Chapter 4**

## **Trademarks**

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

How to Reach Us:

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

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

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

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

© 2017 Freescale Semiconductors, Inc.



# Chapter 5

## ADC: 12-bit Analog to Digital Converter Driver

### 5.1 Overview

The MCUXpresso SDK provides 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

```
volatile bool g_AdcConversionDoneFlag;
volatile uint32_t g_AdcConversionValue;
volatile uint32_t g_AdcInterruptCounter;

// ...

adc_config_t adcConfigStruct;
adc_channel_config_t adcChannelConfigStruct;

ADC_GetDefaultConfig(&adcConfigStruct);
ADC_Init(DEMO_ADC_BASE, &adcConfigStruct);
ADC_EnableHardwareTrigger(DEMO_ADC_BASE, false);
if (kStatus_Success == ADC_DoAutoCalibration(DEMO_ADC_BASE))
{
    PRINTF("ADC_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC_DoAutoCalibration() Failed.\r\n");
}
adcChannelConfigStruct.channelNumber = DEMO_ADC_USER_CHANNEL;
adcChannelConfigStruct.enableInterruptOnConversionCompleted = true;
g_AdcInterruptCounter = 0U;

while (1)
{
    PRINTF("Press any key to get user channel's ADC value.\r\n");
    GETCHAR();
    g_AdcConversionDoneFlag = false;
    ADC_SetChannelConfig(DEMO_ADC_BASE, DEMO_ADC_CHANNEL_GROUP, &adcChannelConfigStruct
        );
    while (g_AdcConversionDoneFlag == false)
    {
    }
    PRINTF("ADC Value: %d\r\n", g_AdcConversionValue);
    PRINTF("ADC Interrupt Counter: %d\r\n", g_AdcInterruptCounter);
}

// ...

void DEMO_ADC_IRQ_HANDLER_FUNC(void)
{
    g_AdcConversionDoneFlag = true;
    g_AdcConversionValue = ADC_GetChannelConversionValue(DEMO_ADC_BASE,
        DEMO_ADC_CHANNEL_GROUP);
    g_AdcInterruptCounter++;
}
```

## Typical use case

}

### 5.2.2 Polling Configuration

```
adc_config_t adcConfigStruct;
adc_channel_config_t adcChannelConfigStruct;

ADC_GetDefaultConfig(&adcConfigStruct);
ADC_Init(DEMO_ADC_BASE, &adcConfigStruct);
ADC_EnableHardwareTrigger(DEMO_ADC_BASE, false);
if (KStatus_Success == ADC_DoAutoCalibration(DEMO_ADC_BASE))
{
    PRINTF("ADC_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC_DoAutoCalibration() Failed.\r\n");
}
adcChannelConfigStruct.channelNumber = DEMO_ADC_USER_CHANNEL;
adcChannelConfigStruct.enableInterruptOnConversionCompleted = false;

while (1)
{
    PRINTF("Press any key to get user channel's ADC value.\r\n");
    GETCHAR();
    ADC_SetChannelConfig(DEMO_ADC_BASE, DEMO_ADC_CHANNEL_GROUP, &adcChannelConfigStruct
        );
    while (0U == ADC_GetChannelStatusFlags(DEMO_ADC_BASE, DEMO_ADC_CHANNEL_GROUP))
    {
    }
    PRINTF("ADC Value: %d\r\n", ADC_GetChannelConversionValue(DEMO_ADC_BASE,
        DEMO_ADC_CHANNEL_GROUP));
}
```

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

## Typical use case

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

## 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**

## Data Structure Documentation

Select the ADC resolution mode.

### 5.3.2 struct adc\_offset\_config\_t

#### Data Fields

- bool enableSigned
  - if false, The offset value is added with the raw result.*
- uint32\_t offsetYValue
  - 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.

## 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\_AsynchronousWakeupInterruptFlag* Asynchronous wakeup interrupt occured, 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.

## Enumeration Type Documentation

*kADC\_ClockSourceIPGDiv2* Select IPG clock divided by 2 to generate ADCK.

*kADC\_ClockSourceALT* Select alternate clock to generate ADCK.

*kADC\_ClockSourceAD* Select Asynchronous clock to generate ADCK.

### 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* )

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 =
* config->enableContinuousConversion =
* config->enableHighSpeed =
* config->enableLowPower =
* config->enableLongSample =
* config->referenceVoltageSource =
* config->samplePeriodMode =
* config->clockSource =
* config->clockDriver =
* config->resolution =
* 
```

Parameters

## Function Documentation

<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.

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.

## Function Documentation

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

Parameters

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

### 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

<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.
-------------	------------------------------

## Variable Documentation

<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".

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`

## Variable Documentation

# Chapter 6

## ADC\_5HC: 12-bit Analog to Digital Converter Driver

### 6.1 Overview

The MCUXpresso SDK provides Peripheral driver for the 12-bit Analog to Digital Converter (ADC\_5HC) module of MCUXpresso SDK devices.

### 6.2 Typical use case

#### 6.2.1 Polling Configuration

```
volatile bool g_AdcConversionDoneFlag;
volatile uint32_t g_AdcConversionValue;
volatile uint32_t g_AdcInterruptCounter;

// ...

adc_5hc_config_t adcConfigStruct;
adc_5hc_channel_config_t adcChannelConfigStruct;

ADC_5HC_GetDefaultConfig(&adcConfigStruct);
ADC_5HC_Init(DEMO_ADC_5HC_BASE, &adcConfigStruct);
ADC_5HC_EnableHardwareTrigger(DEMO_ADC_5HC_BASE, false);
if (kStatus_Success == ADC_5HC_DoAutoCalibration(DEMO_ADC_5HC_BASE))
{
    PRINTF("ADC_5HC_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC_5HC_DoAutoCalibration() Failed.\r\n");
}
adcChannelConfigStruct.channelNumber = DEMO_ADC_5HC_USER_CHANNEL;
adcChannelConfigStruct.enableInterruptOnConversionCompleted = true;
g_AdcInterruptCounter = 0U;

while (1)
{
    PRINTF("Press any key to get user channel's ADC_5HC value.\r\n");
    GETCHAR();
    g_AdcConversionDoneFlag = false;
    ADC_5HC_SetChannelConfig(DEMO_ADC_5HC_BASE, DEMO_ADC_5HC_CHANNEL_GROUP, &
        adcChannelConfigStruct);
    while (g_AdcConversionDoneFlag == false)
    {
    }
    PRINTF("ADC_5HC Value: %d\r\n", g_AdcConversionValue);
    PRINTF("ADC_5HC Interrupt Counter: %d\r\n", g_AdcInterruptCounter);
}

// ...

void DEMO_ADC_5HC IRQ_HANDLER_FUNC(void)
{
    g_AdcConversionDoneFlag = true;
    g_AdcConversionValue = ADC_5HC_GetChannelConversionValue(
        DEMO_ADC_5HC_BASE, DEMO_ADC_5HC_CHANNEL_GROUP);
    g_AdcInterruptCounter++;
}
```

## Typical use case

}

### 6.2.2 Polling Configuration

```
adc_5hc_config_t adcConfigStruct;
adc_5hc_channel_config_t adcChannelConfigStruct;

ADC_5HC_GetDefaultConfig(&adcConfigStruct);
ADC_5HC_Init(DEMO_ADC_5HC_BASE, &adcConfigStruct);
ADC_5HC_EnableHardwareTrigger(DEMO_ADC_5HC_BASE, false);
if (KStatus_Success == ADC_5HC_DoAutoCalibration(DEMO_ADC_5HC_BASE))
{
    PRINTF("ADC_5HC_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC_5HC_DoAutoCalibration() Failed.\r\n");
}
adcChannelConfigStruct.channelNumber = DEMO_ADC_5HC_USER_CHANNEL;
adcChannelConfigStruct.enableInterruptOnConversionCompleted = false;

while (1)
{
    PRINTF("Press any key to get user channel's ADC_5HC value.\r\n");
    GETCHAR();
    ADC_5HC_SetChannelConfig(DEMO_ADC_5HC_BASE, DEMO_ADC_5HC_CHANNEL_GROUP, &
        adcChannelConfigStruct);
    while (0U == ADC_5HC_GetChannelStatusFlags(DEMO_ADC_5HC_BASE,
        DEMO_ADC_5HC_CHANNEL_GROUP))
    {
    }
    PRINTF("ADC_5HC Value: %d\r\n", ADC_5HC_GetChannelConversionValue(
        DEMO_ADC_5HC_BASE, DEMO_ADC_5HC_CHANNEL_GROUP));
}
```

## Data Structures

- struct `adc_5hc_config_t`  
*Converter configuration.* [More...](#)
- struct `adc_5hc_offset_config_t`  
*Converter Offset configuration.* [More...](#)
- struct `adc_5hc_hardware_compare_config_t`  
*ADC hardware compare configuration.* [More...](#)
- struct `adc_5hc_channel_config_t`  
*ADC channel conversion configuration.* [More...](#)

## Macros

- #define `FSL_ADC_5HC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*ADC driver version.*

## Enumerations

- enum `adc_5hc_status_flags_t` {  
 kADC\_5HC\_ConversionActiveFlag = ADC\_5HC\_GS\_ADACT\_MASK,  
 kADC\_5HC\_CalibrationFailedFlag = ADC\_5HC\_GS\_CALF\_MASK,  
 kADC\_5HC\_AsynchronousWakeupInterruptFlag }

- Converter's status flags.
  - enum `adc_5hc_reference_voltage_source_t` { `kADC_5HC_ReferenceVoltageSourceAlt0 = 0U` }
  - Reference voltage source.*
  - enum `adc_5hc_sample_period_mode_t` {
    - `kADC_5HC_SamplePeriod2or12Clocks` = `0U`,
    - `kADC_5HC_SamplePeriod4or16Clocks` = `1U`,
    - `kADC_5HC_SamplePeriod6or20Clocks` = `2U`,
    - `kADC_5HC_SamplePeriod8or24Clocks` = `3U`,
    - `kADC_5HC_SamplePeriodLong12Clclocks` = `kADC_5HC_SamplePeriod2or12Clocks`,
    - `kADC_5HC_SamplePeriodLong16Clclocks` = `kADC_5HC_SamplePeriod4or16Clocks`,
    - `kADC_5HC_SamplePeriodLong20Clclocks` = `kADC_5HC_SamplePeriod6or20Clocks`,
    - `kADC_5HC_SamplePeriodLong24Clclocks` = `kADC_5HC_SamplePeriod8or24Clocks`,
    - `kADC_5HC_SamplePeriodShort2Clocks` = `kADC_5HC_SamplePeriod2or12Clocks`,
    - `kADC_5HC_SamplePeriodShort4Clocks` = `kADC_5HC_SamplePeriod4or16Clocks`,
    - `kADC_5HC_SamplePeriodShort6Clocks` = `kADC_5HC_SamplePeriod6or20Clocks`,
    - `kADC_5HC_SamplePeriodShort8Clocks` = `kADC_5HC_SamplePeriod8or24Clocks` }
  - Sample time duration.*
- enum `adc_5hc_clock_source_t` {
  - `kADC_5HC_ClockSourceIPG` = `0U`,
  - `kADC_5HC_ClockSourceIPGDiv2` = `1U`,
  - `kADC_5HC_ClockSourceAD` = `3U` }
- Clock source.*
- enum `adc_5hc_clock_driver_t` {
  - `kADC_5HC_ClockDriver1` = `0U`,
  - `kADC_5HC_ClockDriver2` = `1U`,
  - `kADC_5HC_ClockDriver4` = `2U`,
  - `kADC_5HC_ClockDriver8` = `3U` }
- Clock divider for the converter.*
- enum `adc_5hc_resolution_t` {
  - `kADC_5HC_Resolution8Bit` = `0U`,
  - `kADC_5HC_Resolution10Bit` = `1U`,
  - `kADC_5HC_Resolution12Bit` = `2U` }
- Converter's resolution.*
- enum `adc_5hc_hardware_compare_mode_t` {
  - `kADC_5HC_HardwareCompareMode0` = `0U`,
  - `kADC_5HC_HardwareCompareMode1` = `1U`,
  - `kADC_5HC_HardwareCompareMode2` = `2U`,
  - `kADC_5HC_HardwareCompareMode3` = `3U` }
- Converter hardware compare mode.*
- enum `adc_5hc_hardware_average_mode_t` {
  - `kADC_5HC_HardwareAverageCount4` = `0U`,
  - `kADC_5HC_HardwareAverageCount8` = `1U`,
  - `kADC_5HC_HardwareAverageCount16` = `2U`,
  - `kADC_5HC_HardwareAverageCount32` = `3U`,
  - `kADC_5HC_HardwareAverageDiasable` = `4U` }
- Converter hardware average mode.*

## Typical use case

### Variables

- bool `adc_5hc_config_t::enableOverWrite`  
*Enable the overwriting.*
- bool `adc_5hc_config_t::enableContinuousConversion`  
*Enable the continuous conversion mode.*
- bool `adc_5hc_config_t::enableHighSpeed`  
*Enable the high-speed mode.*
- bool `adc_5hc_config_t::enableLowPower`  
*Enable the low power mode.*
- bool `adc_5hc_config_t::enableLongSample`  
*Enable the long sample mode.*
- bool `adc_5hc_config_t::enableAsynchronousClockOutput`  
*Enable the asynchronous clock output.*
- `adc_5hc_reference_voltage_source_t adc_5hc_config_t::referenceVoltageSource`  
*Select the reference voltage source.*
- `adc_5hc_sample_period_mode_t adc_5hc_config_t::samplePeriodMode`  
*Select the sample period in long sample mode or short mode.*
- `adc_5hc_clock_source_t adc_5hc_config_t::clockSource`  
*Select the input clock source to generate the internal clock ADCK.*
- `adc_5hc_clock_driver_t adc_5hc_config_t::clockDriver`  
*Select the divide ratio used by the ADC to generate the internal clock ADCK.*
- `adc_5hc_resolution_t adc_5hc_config_t::resolution`  
*Select the ADC resolution mode.*
- bool `adc_5hc_offset_config_t::enableSigned`  
*if false, The offset value is added with the raw result.*
- uint32\_t `adc_5hc_offset_config_t::offsetValue`  
*User configurable offset value(0-4095).*
- `adc_5hc_hardware_compare_mode_t adc_5hc_hardware_compare_config_t::hardwareCompareMode`  
*Select the hardware compare mode.*
- uint16\_t `adc_5hc_hardware_compare_config_t::value1`  
*Setting value1(0-4095) for hardware compare mode.*
- uint16\_t `adc_5hc_hardware_compare_config_t::value2`  
*Setting value2(0-4095) for hardware compare mode.*
- uint32\_t `adc_5hc_channel_config_t::channelNumber`  
*Setting the conversion channel number.*
- bool `adc_5hc_channel_config_t::enableInterruptOnConversionCompleted`  
*Generate an interrupt request once the conversion is completed.*

### Initialization

- void `ADC_5HC_Init (ADC_5HC_Type *base, const adc_5hc_config_t *config)`  
*Initialize the ADC module.*
- void `ADC_5HC_Deinit (ADC_5HC_Type *base)`  
*De-initializes the ADC module.*
- void `ADC_5HC_GetDefaultConfig (adc_5hc_config_t *config)`  
*Gets an available pre-defined settings for the converter's configuration.*
- void `ADC_5HC_SetChannelConfig (ADC_5HC_Type *base, uint32_t channelGroup, const adc_5hc_channel_config_t *config)`  
*Configures the conversion channel.*

- static uint32\_t [ADC\\_5HC\\_GetChannelConversionValue](#) (ADC\_5HC\_Type \*base, uint32\_t channelGroup)  
*Gets the conversion value.*
- static uint32\_t [ADC\\_5HC\\_GetChannelStatusFlags](#) (ADC\_5HC\_Type \*base, uint32\_t channelGroup)  
*Gets the status flags of channel.*
- status\_t [ADC\\_5HC\\_DoAutoCalibration](#) (ADC\_5HC\_Type \*base)  
*Automates the hardware calibration.*
- void [ADC\\_5HC\\_SetOffsetConfig](#) (ADC\_5HC\_Type \*base, const adc\_5hc\_offset\_config\_t \*config)  
*Set user defined offset.*
- static void [ADC\\_5HC\\_EnableDMA](#) (ADC\_5HC\_Type \*base, bool enable)  
*Enables generating the DMA trigger when the conversion is complete.*
- static void [ADC\\_5HC\\_EnableHardwareTrigger](#) (ADC\_5HC\_Type \*base, bool enable)  
*Enables the hardware trigger mode.*
- void [ADC\\_5HC\\_SetHardwareCompareConfig](#) (ADC\_5HC\_Type \*base, const adc\_5hc\_hardware\_compare\_config\_t \*config)  
*Configures the hardware compare mode.*
- void [ADC\\_5HC\\_SetHardwareAverageConfig](#) (ADC\_5HC\_Type \*base, const adc\_5hc\_hardware\_average\_mode\_t mode)  
*Configures the hardware average mode.*
- static uint32\_t [ADC\\_5HC\\_GetStatusFlags](#) (ADC\_5HC\_Type \*base)  
*Gets the converter's status flags.*
- void [ADC\\_5HC\\_ClearStatusFlags](#) (ADC\_5HC\_Type \*base, uint32\_t mask)  
*Clears the converter's status falgs.*

## 6.3 Data Structure Documentation

### 6.3.1 struct adc\_5hc\_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\\_5hc\\_reference\\_voltage\\_source\\_t](#) [referenceVoltageSource](#)  
*Select the reference voltage source.*
- [adc\\_5hc\\_sample\\_period\\_mode\\_t](#) [samplePeriodMode](#)  
*Select the sample period in long sample mode or short mode.*
- [adc\\_5hc\\_clock\\_source\\_t](#) [clockSource](#)  
*Select the input clock source to generate the internal clock ADCK.*
- [adc\\_5hc\\_clock\\_driver\\_t](#) [clockDriver](#)

## Data Structure Documentation

- Select the divide ratio used by the ADC to generate the internal clock ADCK.
- [adc\\_5hc\\_resolution\\_t resolution](#)  
*Select the ADC resolution mode.*

### 6.3.2 struct adc\_5hc\_offset\_config\_t

#### Data Fields

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

### 6.3.3 struct adc\_5hc\_hardware\_compare\_config\_t

In kADC\_5HC\_HardwareCompareMode0, compare true if the result is less than the value1. In kADC\_5HC\_HardwareCompareMode1, compare true if the result is greater than or equal to value1. In kADC\_5HC\_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\_5HC\_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\\_5hc\\_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.*

### 6.3.4 struct adc\_5hc\_channel\_config\_t

#### Data Fields

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

## 6.4 Macro Definition Documentation

### 6.4.1 #define FSL\_ADC\_5HC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

## 6.5 Enumeration Type Documentation

### 6.5.1 enum adc\_5hc\_status\_flags\_t

Enumerator

*kADC\_5HC\_ConversionActiveFlag* Conversion is active,not support w1c.

*kADC\_5HC\_CalibrationFailedFlag* Calibration is failed,support w1c.

*kADC\_5HC\_AsynchronousWakeupInterruptFlag* Asynchronous wakeup interrupt occured, support w1c.

### 6.5.2 enum adc\_5hc\_reference\_voltage\_source\_t

Enumerator

*kADC\_5HC\_ReferenceVoltageSourceAlt0* For external pins pair of VrefH and VrefL.

### 6.5.3 enum adc\_5hc\_sample\_period\_mode\_t

Enumerator

*kADC\_5HC\_SamplePeriod2or12Clocks* Long sample 12 clocks or short sample 2 clocks.

*kADC\_5HC\_SamplePeriod4or16Clocks* Long sample 16 clocks or short sample 4 clocks.

*kADC\_5HC\_SamplePeriod6or20Clocks* Long sample 20 clocks or short sample 6 clocks.

*kADC\_5HC\_SamplePeriod8or24Clocks* Long sample 24 clocks or short sample 8 clocks.

*kADC\_5HC\_SamplePeriodLong12Clcoks* Long sample 12 clocks.

*kADC\_5HC\_SamplePeriodLong16Clcoks* Long sample 16 clocks.

*kADC\_5HC\_SamplePeriodLong20Clcoks* Long sample 20 clocks.

*kADC\_5HC\_SamplePeriodLong24Clcoks* Long sample 24 clocks.

*kADC\_5HC\_SamplePeriodShort2Clocks* Short sample 2 clocks.

*kADC\_5HC\_SamplePeriodShort4Clocks* Short sample 4 clocks.

*kADC\_5HC\_SamplePeriodShort6Clocks* Short sample 6 clocks.

*kADC\_5HC\_SamplePeriodShort8Clocks* Short sample 8 clocks.

## Enumeration Type Documentation

### 6.5.4 enum adc\_5hc\_clock\_source\_t

Enumerator

*kADC\_5HC\_ClockSourceIPG* Select IPG clock to generate ADCK.

*kADC\_5HC\_ClockSourceIPGDiv2* Select IPG clock divided by 2 to generate ADCK.

*kADC\_5HC\_ClockSourceAD* Select Asynchronous clock to generate ADCK.

### 6.5.5 enum adc\_5hc\_clock\_driver\_t

Enumerator

*kADC\_5HC\_ClockDriver1* For divider 1 from the input clock to the module.

*kADC\_5HC\_ClockDriver2* For divider 2 from the input clock to the module.

*kADC\_5HC\_ClockDriver4* For divider 4 from the input clock to the module.

*kADC\_5HC\_ClockDriver8* For divider 8 from the input clock to the module.

### 6.5.6 enum adc\_5hc\_resolution\_t

Enumerator

*kADC\_5HC\_Resolution8Bit* Single End 8-bit resolution.

*kADC\_5HC\_Resolution10Bit* Single End 10-bit resolution.

*kADC\_5HC\_Resolution12Bit* Single End 12-bit resolution.

### 6.5.7 enum adc\_5hc\_hardware\_compare\_mode\_t

Enumerator

*kADC\_5HC\_HardwareCompareMode0* Compare true if the result is less than the value1.

*kADC\_5HC\_HardwareCompareMode1* Compare true if the result is greater than or equal to value1.

*kADC\_5HC\_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\_5HC\_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.

## 6.5.8 enum adc\_5hc.hardware\_average\_mode\_t

Enumerator

- kADC\_5HC\_HardwareAverageCount4* For hardware average with 4 samples.
- kADC\_5HC\_HardwareAverageCount8* For hardware average with 8 samples.
- kADC\_5HC\_HardwareAverageCount16* For hardware average with 16 samples.
- kADC\_5HC\_HardwareAverageCount32* For hardware average with 32 samples.
- kADC\_5HC\_HardwareAverageDiasable* Disable the hardware average function.

## 6.6 Function Documentation

### 6.6.1 void ADC\_5HC\_Init ( ADC\_5HC\_Type \* *base*, const adc\_5hc\_config\_t \* *config* )

Parameters

<i>base</i>	ADC peripheral base address.
<i>config</i>	Pointer to "adc_5hc_config_t" structure.

### 6.6.2 void ADC\_5HC\_Deinit ( ADC\_5HC\_Type \* *base* )

Parameters

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

### 6.6.3 void ADC\_5HC\_GetDefaultConfig ( adc\_5hc\_config\_t \* *config* )

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

```
* config->enableAsynchronousClockOutput = true;
* config->enableOverWrite =
* config->enableContinuousConversion =
* config->enableHighSpeed =
* config->enableLowPower =
* config->enableLongSample =
* config->referenceVoltageSource =
* config->samplePeriodMode =
* config->clockSource =
* config->clockDriver =
* config->resolution =
*
```

## Function Documentation

### Parameters

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

### **6.6.4 void ADC\_5HC\_SetChannelConfig ( ADC\_5HC\_Type \* *base*, uint32\_t *channelGroup*, const adc\_5hc\_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 HCn 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_5hc_channel_config_t" structure for the conversion channel.

### **6.6.5 static uint32\_t ADC\_5HC\_GetChannelConversionValue ( ADC\_5HC\_Type \* *base*, uint32\_t *channelGroup* ) [inline], [static]**

### Parameters

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

Returns

Conversion value.

### 6.6.6 static uint32\_t ADC\_5HC\_GetChannelStatusFlags ( ADC\_5HC\_Type \* *base*, uint32\_t *channelGroup* ) [inline], [static]

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

Parameters

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

Returns

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

### 6.6.7 status\_t ADC\_5HC\_DoAutoCalibration ( ADC\_5HC\_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.

## Function Documentation

Return values

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

**6.6.8 void ADC\_5HC\_SetOffsetConfig ( ADC\_5HC\_Type \* *base*, const adc\_5hc\_offset\_config\_t \* *config* )**

Parameters

<i>base</i>	ADC peripheral base address.
<i>config</i>	Pointer to "adc_5hc_offset_config_t" structure.

**6.6.9 static void ADC\_5HC\_EnableDMA ( ADC\_5HC\_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.

**6.6.10 static void ADC\_5HC\_EnableHardwareTrigger ( ADC\_5HC\_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.

**6.6.11 void ADC\_5HC\_SetHardwareCompareConfig ( ADC\_5HC\_Type \* *base*, const adc\_5hc\_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\_5hc\_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_5hc_hardware_compare_config_t" structure.

### 6.6.12 void ADC\_5HC\_SetHardwareAverageConfig ( ADC\_5HC\_Type \* *base*, adc\_5hc.hardware\_average\_mode\_t *mode* )

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

Parameters

<i>base</i>	ADC peripheral base address.
<i>mode</i>	Setting the hardware average mode. See "adc_5hc.hardware_average_mode_t".

### 6.6.13 static uint32\_t ADC\_5HC\_GetStatusFlags ( ADC\_5HC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Flags' mask if indicated flags are asserted. See "adc\_5hc\_status\_flags\_t".

### 6.6.14 void ADC\_5HC\_ClearStatusFlags ( ADC\_5HC\_Type \* *base*, uint32\_t *mask* )

Parameters

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

## Variable Documentation

<i>mask</i>	Mask value for the cleared flags. See "adc_5hc_status_flags_t".
-------------	---

### 6.7 Variable Documentation

6.7.1 **bool adc\_5hc\_config\_t::enableOverWrite**

6.7.2 **bool adc\_5hc\_config\_t::enableContinuousConversion**

6.7.3 **bool adc\_5hc\_config\_t::enableHighSpeed**

6.7.4 **bool adc\_5hc\_config\_t::enableLowPower**

6.7.5 **bool adc\_5hc\_config\_t::enableLongSample**

6.7.6 **bool adc\_5hc\_config\_t::enableAsynchronousClockOutput**

6.7.7 **adc\_5hc\_reference\_voltage\_source\_t adc\_5hc\_config\_t::referenceVoltageSource**

6.7.8 **adc\_5hc\_sample\_period\_mode\_t adc\_5hc\_config\_t::samplePeriodMode**

6.7.9 **adc\_5hc\_clock\_source\_t adc\_5hc\_config\_t::clockSource**

6.7.10 **adc\_5hc\_clock\_driver\_t adc\_5hc\_config\_t::clockDriver**

6.7.11 **adc\_5hc\_resolution\_t adc\_5hc\_config\_t::resolution**

6.7.12 **bool adc\_5hc\_offset\_config\_t::enableSigned**

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

6.7.13 **uint32\_t adc\_5hc\_offset\_config\_t::offsetValue**

6.7.14 **adc\_5hc\_hardware\_compare\_mode\_t adc\_5hc\_hardware\_compare\_config\_t::hardwareCompareMode**

See "adc\_5hc\_hardware\_compare\_mode\_t".

6.7.15 `uint16_t adc_5hc.hardware_compare_config_t::value1`

6.7.16 `uint16_t adc_5hc.hardware_compare_config_t::value2`

6.7.17 `uint32_t adc_5hc.channel_config_t::channelNumber`

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

6.7.18 `bool adc_5hc.channel_config_t::enableInterruptOnConversionCompleted`

## Variable Documentation

# Chapter 7

## CACHE: CACHE Memory Controller

### 7.1 Overview

The MCUXpresso SDK provides Peripheral driver for the CACHE Controller.

CACHE driver is created to help user to operate the Cache memory more easy. The APIs for basic operations are including the following two levels: 1L. The L1 cache driver API. The L1 cache driver API here is a defined as core integrated caches controller driver for all core-A intergreated caches: a. L1 maintain only if L2 is not supported b. L1 and L2 cache maintain together.

2L. The combined cache driver API. This level provides unified APIs for combined cache maintain operations. This is provided for MCUXpresso SDK drivers (DMA, ENET, USDHC etc) which should do the cache maintenance in their transactional APIs. In this architecture, there is only core intergrated cache, so the unified APIs here is directly call the L1 cache driver API.

### 7.2 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.

#### Driver version

- #define `FSL_CACHE_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*cache driver version 2.0.0.*

#### Cache Control for Cortex-a L1 cache

- static void `L1CACHE_EnableICache` (void)  
*Enables L1 instruction cache.*
- static void `L1CACHE_DisableICache` (void)  
*Disables L1 instruction cache.*
- static void `L1CACHE_InvalidateICache` (void)  
*Invalidate L1 instruction cache all.*
- static void `L1CACHE_InvalidateICacheByRange` (uint32\_t startAddr, uint32\_t size\_byte)  
*Invalidate L1 instruction cache by range.*
- static void `L1CACHE_EnableDCache` (void)  
*Enables L1 data cache.*
- static void `L1CACHE_DisableDCache` (void)  
*Disables L1 data cache.*
- static void `L1CACHE_InvalidateDCache` (void)

## Function Documentation

- static void [L1CACHE\\_InvalidateDCacheByRange](#) (uint32\_t startAddr, uint32\_t size\_byte)  
*Invalidates L1 data cache by range.*
- static void [L1CACHE\\_CleanDCache](#) (void)  
*Clean L1 data cache all.*
- static void [L1CACHE\\_CleanDCacheByRange](#) (uint32\_t startAddr, uint32\_t size\_byte)  
*Cleans L1 data cache by range.*
- static void [L1CACHE\\_CleanInvalidateDCache](#) (void)  
*Cleans and invalidates L1 data cache all.*
- static void [L1CACHE\\_CleanInvalidateDCacheByRange](#) (uint32\_t startAddr, uint32\_t size\_byte)  
*Cleans and invalidates L1 data cache by range.*

## Unified Cache Control for all caches which is mainly used for

SDK Driver easy use cache driver

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

## 7.3 Macro Definition Documentation

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

## 7.4 Function Documentation

### 7.4.1 static void L1CACHE\_InvalidateCacheByRange ( uint32\_t startAddr, uint32\_t size\_byte ) [inline], [static]

Parameters

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

Note

The start startAddr and size\_byte should be 32-byte(FSL\_FEATURE\_L1ICACHE\_LINESIZE\_BY-TE) aligned due to the cache operation unit is one L1 I-cache line. 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.

7.4.2 **static void L1CACHE\_InvalidateDCacheByRange ( uint32\_t *startAddr*,  
          uint32\_t *size\_byte* ) [inline], [static]**

## Function Documentation

Parameters

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

Note

The start startAddr and size\_byte should be 64-byte(FSL FEATURE\_L1DCACHE\_LINESIZE\_BYTEx) aligned due to the cache operation unit is one L1 D-cache line. 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.

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

Parameters

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

Note

The start startAddr and size\_byte should be 64-byte(FSL FEATURE\_L1DCACHE\_LINESIZE\_BYTEx) aligned due to the cache operation unit is one L1 D-cache line. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

### 7.4.4 static void L1CACHE\_CleanInvalidateDCacheByRange ( uint32\_t *startAddr*, uint32\_t *size\_byte* ) [inline], [static]

Parameters

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

Note

The start startAddr and size\_byte should be 64-byte(FSL FEATURE\_L1DCACHE\_LINESIZE\_BYTEx) aligned due to the cache operation unit is one L1 D-cache line. The startAddr here will be forced to align to L1 D-cache line size if startAddr is not aligned. For size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

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

Cortex-a L1 instruction cache line length is 32-byte.

## Function Documentation

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.

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

Cortex-a L1 data cache line length is 64-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 64-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.

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

Cortex-a L1 data cache line length is 64-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 64-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.

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

Cortex-a L1 data cache line length is 64-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 64-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.

## Function Documentation

# Chapter 8

## CSI: CMOS Sensor Interface

### 8.1 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 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.

### 8.2 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 re-defining 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.

### 8.3 Typical use case

#### 8.3.1 Receive with functional APIs

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

```
uint32_t frameBuffers[10][320][240];
```

## Typical use case

```
volatile uint32_t frameIdx = 0;
volatile uint32_t frameRecv = 0;

void CSI_IRQHandler(void)
{
    uint32_t intStat = CSI_GetStatusFlags(CSI);

    if (frameIdx < 10)
    {
        if (intStat & kCSI_RxBuffer0DmaDoneFlag)
        {
            CSI_SetRxBufferAddr(CSI, 0, (uint32_t)frameBuffer[frameIdx++]);
        }

        if (intStat & kCSI_RxBuffer1DmaDoneFlag)
        {
            CSI_SetRxBufferAddr(CSI, 1, (uint32_t)frameBuffer[frameIdx++]);
        }
    }

    CSI_ClearStatusFlags(CSI, kCSI_RxBuffer0DmaDoneFlag | kCSI_RxBuffer1DmaDoneFlag);

    frameRecv++;
}

void main(void)
{
    /* Initialize the CSI first. */
    const csi_config_t csiConfig =
    {
        // ...
    };

    CSI_Init(CSI, &csiConfig);

    CSI_EnableInterrupts(CSI,
        kCSI_RxBuffer1DmaDoneInterruptEnable |
        kCSI_RxBuffer0DmaDoneInterruptEnable);

    /* Set frame buffer address to CSI. */
    CSI_SetRxBufferAddr(CSI, 0, (uint32_t)frameBuffer[frameIdx++]);
    CSI_SetRxBufferAddr(CSI, 1, (uint32_t)frameBuffer[frameIdx++]);

    CSI_Start(TEST_CSI);

    while (frameRecv < 10)
    {

    }

    CSI_DisableInterrupts(CSI,
        kCSI_RxBuffer1DmaDoneInterruptEnable |
        kCSI_RxBuffer0DmaDoneInterruptEnable);
    CSI_Stop(TEST_CSI);
}
```

### 8.3.2 Receive with transactional APIs

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

```
#define FRAME_BUFFER_CNT 3

uint32_t frameBuffers[FRAME_BUFFER_CNT][320][240];
csi_handle_t handle;
```

```

uint32_t bufferToProcess;

void main(void)
{
    uint32_t i;

    /* Initialize the CSI first. */
    const csi_config_t csiConfig =
    {
        // ...
    };

    CSI_Init(CSI, &csiConfig);

    /* Create the handle. */
    CSI_TransferCreateHandle(CSI, &handle, NULL, NULL);

    /* Put the buffer to the queue. */
    for (i=0; i<FRAME_BUFFER_CNT; i++)
    {
        CSI_TransferSubmitEmptyBuffer(CSI, &handle, (uint32_t)(frameBuffers[i])
        ));
    }

    /* Start transfer. */
    CSI_TransferStart(CSI, &handle);

    while (1)
    {
        /* Wait to get the full frame buffer. */
        while (kStatus_Success != CSI_TransferGetFullBuffer(CSI, &handle, &
bufferToProcess))
        {
        }

        /* Now bufferToProcess points to the full-filled buffer, could start process. */
        // ...

        /* Process finished, put the buffer back to the queue. */
        CSI_TransferSubmitEmptyBuffer(CSI, &handle, bufferToProcess);
    }
}

```

In this example, there is 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.*

## Typical use case

## 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 }`  
*CSI data bus width.*
- `enum _csi_polarity_flags {`  
    `kCSI_HsyncActiveLow = 0U,`  
    `kCSI_HsyncActiveHigh = CSI_CSICR1_HSYNC_POL_MASK,`  
    `kCSI_DataLatchOnRisingEdge = CSI_CSICR1_REDGE_MASK,`  
    `kCSI_DataLatchOnFallingEdge = 0U }`  
*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_Field0DoneInterruptEnable = CSI_CSICR18_FIELD0_DONE_IE_MASK << 8U,`  
    `kCSI_Field1DoneInterruptEnable = CSI_CSICR18_DMA_FIELD1_DONE_IE_MASK << 8U }`

- *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.*

## 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_RefleshFifoDma (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)`

## Data Structure Documentation

- void [CSI\\_SetRxBufferAddr](#) (CSI\_Type \*base, uint8\_t index, uint32\_t addr)  
*Stop to receiving data.  
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 frameBuffer)  
*Submit empty frame buffer to queue.*
- status\_t [CSI\\_TransferGetFullBuffer](#) (CSI\_Type \*base, csi\_handle\_t \*handle, uint32\_t \*frameBuffer)  
*Get one full frame buffer from queue.*
- void [CSI\\_TransferHandleIRQ](#) (CSI\_Type \*base, csi\_handle\_t \*handle)  
*CSI IRQ handle function.*

## 8.4 Data Structure Documentation

### 8.4.1 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.*

#### 8.4.1.0.0.1 Field Documentation

##### 8.4.1.0.0.1.1 `uint16_t csi_config_t::width`

##### 8.4.1.0.0.1.2 `uint16_t csi_config_t::height`

##### 8.4.1.0.0.1.3 `uint32_t csi_config_t::polarityFlags`

##### 8.4.1.0.0.1.4 `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.

##### 8.4.1.0.0.1.5 `uint16_t csi_config_t::linePitch_Bytes`

##### 8.4.1.0.0.1.6 `csi_work_mode_t csi_config_t::workMode`

##### 8.4.1.0.0.1.7 `csi_data_bus_t csi_config_t::dataBus`

##### 8.4.1.0.0.1.8 `bool csi_config_t::useExtVsync`

#### 8.4.2 `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.*

## Typedef Documentation

- 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.*

### 8.4.2.0.0.2 Field Documentation

8.4.2.0.0.2.1 **uint32\_t csi\_handle\_t::frameBufferQueue[CSI\_DRIVER\_ACTUAL\_QUEUE\_SIZE]**

8.4.2.0.0.2.2 **volatile uint8\_t csi\_handle\_t::queueUserReadIdx**

8.4.2.0.0.2.3 **volatile uint8\_t csi\_handle\_t::queueUserWriteIdx**

8.4.2.0.0.2.4 **volatile uint8\_t csi\_handle\_t::queueDrvReadIdx**

8.4.2.0.0.2.5 **volatile uint8\_t csi\_handle\_t::queueDrvWriteIdx**

8.4.2.0.0.2.6 **volatile uint8\_t csi\_handle\_t::activeBufferNum**

8.4.2.0.0.2.7 **volatile uint8\_t csi\_handle\_t::nextBufferIdx**

8.4.2.0.0.2.8 **volatile bool csi\_handle\_t::transferStarted**

8.4.2.0.0.2.9 **volatile bool csi\_handle\_t::transferOnGoing**

8.4.2.0.0.2.10 **csi\_transfer\_callback\_t csi\_handle\_t::callback**

8.4.2.0.0.2.11 **void\* csi\_handle\_t::userData**

## 8.5 Macro Definition Documentation

8.5.1 **#define CSI\_DRIVER\_QUEUE\_SIZE 4U**

## 8.6 Typedef Documentation

8.6.1 **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.

## 8.7 Enumeration Type Documentation

### 8.7.1 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.

### 8.7.2 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.

### 8.7.3 enum csi\_data\_bus\_t

Currently only support 8-bit width.

Enumerator

*kCSI\_DataBus8Bit* 8-bit data bus.

### 8.7.4 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.

## Enumeration Type Documentation

### 8.7.5 enum csi\_fifo\_t

Enumerator

*kCSI\_RxFifo* RXFIFO.

*kCSI\_StatFifo* STAT FIFO.

*kCSI\_AllFifo* Both RXFIFO and STAT FIFO.

### 8.7.6 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\_Field0DoneInterruptEnable* Field 0 done interrupt enable.

*kCSI\_Field1DoneInterruptEnable* Field 1 done interrupt enable.

### 8.7.7 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.  
*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.

## 8.8 Function Documentation

### 8.8.1 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.

### 8.8.2 void CSI\_Deinit ( CSI\_Type \* *base* )

This function disables the CSI peripheral clock.

## Function Documentation

Parameters

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

### 8.8.3 void CSI\_Reset ( *CSI\_Type* \* *base* )

This function resets the CSI peripheral registers to default status.

Parameters

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

### 8.8.4 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.
---------------	-----------------------------------

### 8.8.5 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.
-------------	--------------------

### 8.8.6 void CSI\_RefleshFifoDma ( **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.

Parameters

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

### 8.8.7 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.

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

Parameters

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

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

## Function Documentation

Parameters

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

### 8.8.10 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.

### 8.8.11 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> .

### 8.8.12 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> .

### 8.8.13 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](#).

### 8.8.14 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](#),
- [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> .

### 8.8.15 **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.
------------------------	------------------------------

### 8.8.16 **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.

## Function Documentation

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.

### 8.8.17 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>	Stopped successfully.
------------------------	-----------------------

### 8.8.18 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.
<i>kStatus_CSI_QueueFull</i>	Could not submit because there is no room in queue.

### 8.8.19 **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.

### 8.8.20 **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.

## Function Documentation

# **Chapter 9**

## **ECSPI: Serial Peripheral Interface Driver**

### **9.1 Overview**

#### **Modules**

- ECSPI Driver
- ECSPI FreeRTOS Driver

### 9.2 ECSPi Driver

#### 9.2.1 Overview

ECSPi driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for ecSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. ECSPi functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the spi\_handle\_t as the first parameter. Initialize the handle by calling the SPI\_MasterTransferCreateHandle() or SPI\_SlaveTransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SPI\_MasterTransferNonBlocking() and SPI\_SlaveTransferNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus\_SPI\_Idle status.

#### 9.2.2 Typical use case

##### 9.2.2.1 SPI master transfer using polling method

```
#define TRANSFER_SIZE 64U           /* Transfer dataSize */

uint32_t masterRxData[TRANSFER_SIZE] = {0U};
uint32_t masterTxData[TRANSFER_SIZE] = {0U};

void main(void)
{
    uint32_t srcClock_Hz;
    uint32_t errorCount;
    uint32_t loopCount = 1U;
    ecspi_master_config_t masterConfig;
    ecspi_transfer_t masterXfer;

    /* Master config */
    ECSPi_MasterGetDefaultConfig(&masterConfig);
    srcClock_Hz = CLOCK_GetFreq(ECSPi_MASTER_CLK_SRC);
    ECSPi_MasterInit(EXAMPLE_ECSPi_MASTER_BASEADDR, &masterConfig, srcClock_Hz);

    while (1)
    {
        /* Start master transfer, send data to slave */
        masterXfer.txData = masterTxData;
        masterXfer.rxData = NULL;
        masterXfer.dataSize = TRANSFER_SIZE;
        masterXfer.channel = kECSPi_Channel0;
        ECSPi_MasterTransferBlocking(EXAMPLE_ECSPi_MASTER_BASEADDR, &masterXfer
    });

    /* Delay to wait slave is ready */
    for (i = 0U; i < EXAMPLE_ECSPi_DEALY_COUNT; i++)
}
```

```

    {
        __NOP();
    }

    /* Start master transfer, receive data from slave */
    masterXfer.txData = NULL;
    masterXfer.rxData = masterRxData;
    masterXfer.dataSize = TRANSFER_SIZE;
    masterXfer.channel = kECSPI_Channel10;
    ECSPI_MasterTransferBlocking(EXAMPLE_ECSPI_MASTER_BASEADDR, &masterXfer
);
}

/* Wait for press any key */
PRINTF("\r\n Press any key to run again\r\n");
GETCHAR();

/* Increase loop count to change transmit buffer */
loopCount++;
}
}

```

### 9.2.2.2 SPI master transfer using an interrupt method

```

#define BUFFER_LEN (64)
ecspi_master_handle_t spiHandle;
ecspi_master_config_t masterConfig;
ecspi_transfer_t xfer;
volatile bool isFinished = false;

const uint32_t sendData[BUFFER_LEN] = [.....];
uint32_t receiveBuff[BUFFER_LEN];

void ECSPI_UserCallback(ECSPI_Type *base, ecspi_master_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    ECSPI_MasterGetDefaultConfig(&masterConfig);

    ECSPI_MasterInit(ECSP4, &masterConfig);
    ECSPI_MasterTransferCreateHandle(ECSP4, &ecspiHandle,
        ECSPI_UserCallback, NULL);

    // Prepare to send.
    xfer.txData = sendData;
    xfer.rxData = receiveBuff;
    xfer.dataSize = BUFFER_LEN;

    // Send out.
    ECSPI_MasterTransferNonBlocking(ECSP4, &ecspiHandle, &xfer);

    // Wait send finished.
    while (!isFinished)
    {

    }
}

// ...
}

```

## ECSPI Driver

### Data Structures

- struct `ecspi_channel_config_t`  
*ECSPI user channel configure structure.* [More...](#)
- struct `ecspi_master_config_t`  
*ECSPI master configure structure.* [More...](#)
- struct `ecspi_slave_config_t`  
*ECSPI slave configure structure.* [More...](#)
- struct `ecspi_transfer_t`  
*ECSPI transfer structure.* [More...](#)
- struct `ecspi_master_handle_t`  
*ECSPI master handle structure.* [More...](#)

### Macros

- #define `ECSPI_DUMMYDATA` (0xFFFFFFFFU)  
*ECSPI dummy transfer data, the data is sent while txBuff is NULL.*

### Typedefs

- typedef `ecspi_master_handle_t ecspi_slave_handle_t`  
*Slave handle is the same with master handle.*
- typedef void(\* `ecspi_master_callback_t` )(ECSPI\_Type \*base, `ecspi_master_handle_t` \*handle, `status_t` status, void \*userData)  
*ECSPI master callback for finished transmit.*
- typedef void(\* `ecspi_slave_callback_t` )(ECSPI\_Type \*base, `ecspi_slave_handle_t` \*handle, `status_t` status, void \*userData)  
*ECSPI slave callback for finished transmit.*

### Enumerations

- enum `_ecspi_status` {  
  `kStatus_ECSPI_Busy` = MAKE\_STATUS(kStatusGroup\_ECSPI, 0),  
  `kStatus_ECSPI_Idle` = MAKE\_STATUS(kStatusGroup\_ECSPI, 1),  
  `kStatus_ECSPI_Error` = MAKE\_STATUS(kStatusGroup\_ECSPI, 2),  
  `kStatus_ECSPI_HardwareOverFlow` = MAKE\_STATUS(kStatusGroup\_ECSPI, 3) }  
    *Return status for the ECSPI driver.*
- enum `ecspi_clock_polarity_t` {  
  `kECSPI_PolarityActiveHigh` = 0x0U,  
  `kECSPI_PolarityActiveLow` }  
    *ECSPI clock polarity configuration.*
- enum `ecspi_clock_phase_t` {  
  `kECSPI_ClockPhaseFirstEdge`,  
  `kECSPI_ClockPhaseSecondEdge` }  
    *ECSPI clock phase configuration.*

- enum `_ecspi_interrupt_enable` {
   
kECSPI\_Tx\_fifoEmptyInterruptEnable = ECSPI\_INTREG\_TEEN\_MASK,
   
kECSPI\_Tx\_FifoDataRequestInterruptEnable = ECSPI\_INTREG\_TDREN\_MASK,
   
kECSPI\_Tx\_FifoFullInterruptEnable = ECSPI\_INTREG\_TFEN\_MASK,
   
kECSPI\_Rx\_FifoReadyInterruptEnable = ECSPI\_INTREG\_RREN\_MASK,
   
kECSPI\_Rx\_FifoDataRequestInterruptEnable = ECSPI\_INTREG\_RDREN\_MASK,
   
kECSPI\_Rx\_FifoFullInterruptEnable = ECSPI\_INTREG\_RFEN\_MASK,
   
kECSPI\_Rx\_FifoOverflowInterruptEnable = ECSPI\_INTREG\_ROEN\_MASK,
   
kECSPI\_TransferCompleteInterruptEnable = ECSPI\_INTREG\_TCEN\_MASK,
   
kECSPI\_AllInterruptEnable }

*ECSPI interrupt sources.*

- enum `_ecspi_flags` {
   
kECSPI\_Tx\_fifoEmptyFlag = ECSPI\_STATREG\_TE\_MASK,
   
kECSPI\_Tx\_FifoDataRequestFlag = ECSPI\_STATREG\_TDR\_MASK,
   
kECSPI\_Tx\_FifoFullFlag = ECSPI\_STATREG\_TF\_MASK,
   
kECSPI\_Rx\_FifoReadyFlag = ECSPI\_STATREG\_RR\_MASK,
   
kECSPI\_Rx\_FifoDataRequestFlag = ECSPI\_STATREG\_RDR\_MASK,
   
kECSPI\_Rx\_FifoFullFlag = ECSPI\_STATREG\_RF\_MASK,
   
kECSPI\_Rx\_FifoOverflowFlag = ECSPI\_STATREG\_RO\_MASK,
   
kECSPI\_TransferCompleteFlag = ECSPI\_STATREG\_TC\_MASK }

*ECSPI status flags.*

- enum `_ecspi_dma_enable_t` {
   
kECSPI\_TxDmaEnable = ECSPI\_DMAREG\_TEDEN\_MASK,
   
kECSPI\_RxDmaEnable = ECSPI\_DMAREG\_RXDEN\_MASK,
   
kECSPI\_DmaAllEnable = (ECSPI\_DMAREG\_TEDEN\_MASK | ECSPI\_DMAREG\_RXDEN\_MASK) }

*ECSPI DMA enable.*

- enum `ecspi_Data_ready_t` {
   
kECSPI\_DataReadyIgnore = 0x0U,
   
kECSPI\_DataReadyFallingEdge,
   
kECSPI\_DataReadyLowLevel }

*ECSPI SPI\_RDY signal configuration.*

- enum `ecspi_channel_source_t` {
   
kECSPI\_Channel0 = 0x0U,
   
kECSPI\_Channel1,
   
kECSPI\_Channel2,
   
kECSPI\_Channel3 }

*ECSPI channel select source.*

- enum `ecspi_master_slave_mode_t` {
   
kECSPI\_Slave = 0U,
   
kECSPI\_Master }

*ECSPI master or slave mode configuration.*

- enum `ecspi_data_line_inactive_state_t` {
   
kECSPI\_DataLineInactiveStateHigh = 0x0U,
   
kECSPI\_DataLineInactiveStateLow }

*ECSPI data line inactive state configuration.*

## ECSPI Driver

- enum `ecspi_clock_inactive_state_t` {  
  `kECSPI_ClockInactiveStateLow` = 0x0U,  
  `kECSPI_ClockInactiveStateHigh` }  
    *ECSPI clock inactive state configuration.*
- enum `ecspi_chip_select_active_state_t` {  
  `kECSPI_ChipSelectActiveStateLow` = 0x0U,  
  `kECSPI_ChipSelectActiveStateHigh` }  
    *ECSPI active state configuration.*
- enum `ecspi_wave_form_t` {  
  `kECSPI_WaveFormSingle` = 0x0U,  
  `kECSPI_WaveFormMultiple` }  
    *ECSPI wave form configuration.*
- enum `ecspi_sample_period_clock_source_t` {  
  `kECSPI_spiClock` = 0x0U,  
  `kECSPI_lowFreqClock` }  
    *ECSPI sample period clock configuration.*

## Driver version

- #define `FSL_ECSPI_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
    *ECSPI driver version 2.0.0.*

## Initialization and deinitialization

- void `ECSPI_MasterGetDefaultConfig` (`ecspi_master_config_t` \*config)  
      *Sets the ECSPI configuration structure to default values.*
- void `ECSPI_MasterInit` (`ECSPI_Type` \*base, const `ecspi_master_config_t` \*config, `uint32_t` src-Clock\_Hz)  
      *Initializes the ECSPI with configuration.*
- void `ECSPI_SlaveGetDefaultConfig` (`ecspi_slave_config_t` \*config)  
      *Sets the ECSPI configuration structure to default values.*
- void `ECSPI_SlaveInit` (`ECSPI_Type` \*base, const `ecspi_slave_config_t` \*config)  
      *Initializes the ECSPI with configuration.*
- void `ECSPI_Deinit` (`ECSPI_Type` \*base)  
      *De-initializes the ECSPI.*
- static void `ECSPI_Enable` (`ECSPI_Type` \*base, bool enable)  
      *Enables or disables the ECSPI.*

## Status

- static `uint32_t` `ECSPI_GetStatusFlags` (`ECSPI_Type` \*base)  
      *Gets the status flag.*
- static void `ECSPI_ClearStatusFlags` (`ECSPI_Type` \*base, `uint32_t` mask)  
      *Clear the status flag.*

## Interrupts

- static void **ECSPI\_EnableInterrupts** (ECSPI\_Type \*base, uint32\_t mask)  
*Enables the interrupt for the ECSPi.*
- static void **ECSPI\_DisableInterrupts** (ECSPI\_Type \*base, uint32\_t mask)  
*Disables the interrupt for the ECSPi.*

## Software Reset

- static void **ECSPI\_SoftwareReset** (ECSPI\_Type \*base)  
*Software reset.*

## Channel mode check

- static bool **ECSPI\_IsMaster** (ECSPI\_Type \*base, **ecspi\_channel\_source\_t** channel)  
*Mode check.*

## DMA Control

- static void **ECSPI\_EnableDMA** (ECSPI\_Type \*base, uint32\_t mask, bool enable)  
*Enables the DMA source for ECSPi.*

## FIFO Operation

- static uint8\_t **ECSPI\_GetTxFifoCount** (ECSPI\_Type \*base)  
*Get the Tx FIFO data count.*
- static uint8\_t **ECSPI\_GetRxFifoCount** (ECSPI\_Type \*base)  
*Get the Rx FIFO data count.*

## Bus Operations

- static void **ECSPI\_SetChannelSelect** (ECSPI\_Type \*base, **ecspi\_channel\_source\_t** channel)  
*Set channel select for transfer.*
- void **ECSPI\_SetChannelConfig** (ECSPI\_Type \*base, **ecspi\_channel\_source\_t** channel, const **ecspi\_channel\_config\_t** \*config)  
*Set channel select configuration for transfer.*
- void **ECSPI\_SetBaudRate** (ECSPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the baud rate for ECSPi transfer.*
- void **ECSPI\_WriteBlocking** (ECSPI\_Type \*base, uint32\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes using a blocking method.*
- void **ECSPI\_WriteData** (ECSPI\_Type \*base, uint32\_t data)  
*Writes a data into the ECSPi data register.*
- uint32\_t **ECSPI\_ReadData** (ECSPI\_Type \*base)  
*Gets a data from the ECSPi data register.*

### Transactional

- void [`ECSPI\_MasterTransferCreateHandle`](#) (`EC SPI_Type *base`, `ecspi_master_handle_t *handle`,  
`ecspi_master_callback_t callback`, `void *userData`)  
    *Initializes the EC SPI master handle.*
- status\_t [`ECSPI\_MasterTransferBlocking`](#) (`EC SPI_Type *base`, `ecspi_transfer_t *xfer`)  
    *Transfers a block of data using a polling method.*
- status\_t [`ECSPI\_MasterTransferNonBlocking`](#) (`EC SPI_Type *base`, `ecspi_master_handle_t *handle`,  
`ecspi_transfer_t *xfer`)  
    *Performs a non-blocking EC SPI interrupt transfer.*
- status\_t [`ECSPI\_MasterTransferGetCount`](#) (`EC SPI_Type *base`, `ecspi_master_handle_t *handle`,  
`size_t *count`)  
    *Gets the bytes of the EC SPI interrupt transferred.*
- void [`ECSPI\_MasterTransferAbort`](#) (`EC SPI_Type *base`, `ecspi_master_handle_t *handle`)  
    *Aborts an EC SPI transfer using interrupt.*
- void [`ECSPI\_MasterTransferHandleIRQ`](#) (`EC SPI_Type *base`, `ecspi_master_handle_t *handle`)  
    *Interrupts the handler for the EC SPI.*
- void [`ECSPI\_SlaveTransferCreateHandle`](#) (`EC SPI_Type *base`, `ecspi_slave_handle_t *handle`, `ecspi_slave_callback_t callback`, `void *userData`)  
    *Initializes the EC SPI slave handle.*
- static status\_t [`ECSPI\_SlaveTransferNonBlocking`](#) (`EC SPI_Type *base`, `ecspi_slave_handle_t *handle`,  
`ecspi_transfer_t *xfer`)  
    *Performs a non-blocking EC SPI slave interrupt transfer.*
- static status\_t [`ECSPI\_SlaveTransferGetCount`](#) (`EC SPI_Type *base`, `ecspi_slave_handle_t *handle`,  
`size_t *count`)  
    *Gets the bytes of the EC SPI interrupt transferred.*
- static void [`ECSPI\_SlaveTransferAbort`](#) (`EC SPI_Type *base`, `ecspi_slave_handle_t *handle`)  
    *Aborts an EC SPI slave transfer using interrupt.*
- void [`ECSPI\_SlaveTransferHandleIRQ`](#) (`EC SPI_Type *base`, `ecspi_slave_handle_t *handle`)  
    *Interrupts a handler for the EC SPI slave.*

### 9.2.3 Data Structure Documentation

#### 9.2.3.1 struct `ecspi_channel_config_t`

##### Data Fields

- `ecspi_master_slave_mode_t channelMode`  
    *Channel mode.*
- `ecspi_clock_inactive_state_t clockInactiveState`  
    *Clock line (SCLK) inactive state.*
- `ecspi_data_line_inactive_state_t dataLineInactiveState`  
    *Data line (MOSI&MISO) inactive state.*
- `ecspi_chip_select_active_state_t chipSelectActiveState`  
    *Chip select(SS) line active state.*
- `ecspi_wave_form_t waveForm`  
    *Wave form.*
- `ecspi_clock_polarity_t polarity`

- *Clock polarity.*
- `ecspi_clock_phase_t` `phase`  
*Clock phase.*

### 9.2.3.2 `struct ecspi_master_config_t`

#### Data Fields

- `ecspi_channel_source_t` `channel`  
*Channel number.*
- `ecspi_channel_config_t` `channelConfig`  
*Channel configuration.*
- `ecspi_sample_period_clock_source_t` `samplePeriodClock`  
*Sample period clock source.*
- `uint8_t` `burstLength`  
*Burst length.*
- `uint8_t` `chipSelectDelay`  
*SS delay time.*
- `uint16_t` `samplePeriod`  
*Sample period.*
- `uint8_t` `txFifoThreshold`  
*TX Threshold.*
- `uint8_t` `rxFifoThreshold`  
*RX Threshold.*
- `uint32_t` `baudRate_Bps`  
*ECSPI baud rate for master mode.*

### 9.2.3.3 `struct ecspi_slave_config_t`

#### Data Fields

- `uint8_t` `burstLength`  
*Burst length.*
- `uint8_t` `txFifoThreshold`  
*TX Threshold.*
- `uint8_t` `rxFifoThreshold`  
*RX Threshold.*
- `ecspi_channel_config_t` `channelConfig`  
*Channel configuration.*

### 9.2.3.4 `struct ecspi_transfer_t`

#### Data Fields

- `uint32_t *` `txData`  
*Send buffer.*
- `uint32_t *` `rxData`  
*Receive buffer.*

## ECSPI Driver

- `size_t dataSize`  
*Transfer bytes.*
- `ecspi_channel_source_t channel`  
*ECSPI channel select.*

### 9.2.3.5 `struct _ecspi_master_handle`

#### Data Fields

- `ecspi_channel_source_t channel`  
*Channel number.*
- `uint32_t *volatile txData`  
*Transfer buffer.*
- `uint32_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t txRemainingBytes`  
*Send data remaining in bytes.*
- `volatile size_t rxRemainingBytes`  
*Receive data remaining in bytes.*
- `volatile uint32_t state`  
*ECSPI internal state.*
- `size_t transferSize`  
*Bytes to be transferred.*
- `ecspi_master_callback_t callback`  
*ECSPI callback.*
- `void *userData`  
*Callback parameter.*

### 9.2.4 Macro Definition Documentation

#### 9.2.4.1 `#define FSL_ECSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

#### 9.2.4.2 `#define ECSPI_DUMMYDATA (0xFFFFFFFFU)`

### 9.2.5 Enumeration Type Documentation

#### 9.2.5.1 `enum _ecspi_status`

Enumerator

- `kStatus_ECSPI_Busy` ECSPI bus is busy.
- `kStatus_ECSPI_Idle` ECSPI is idle.
- `kStatus_ECSPI_Error` ECSPI error.
- `kStatus_ECSPI_HardwareOverflow` ECSPI hardware overflow.

### 9.2.5.2 enum ecspi\_clock\_polarity\_t

Enumerator

*kECSPI\_PolarityActiveHigh* Active-high ECSPI polarity high (idles low).

*kECSPI\_PolarityActiveLow* Active-low ECSPI polarity low (idles high).

### 9.2.5.3 enum ecspi\_clock\_phase\_t

Enumerator

*kECSPI\_ClockPhaseFirstEdge* First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

*kECSPI\_ClockPhaseSecondEdge* First edge on SPSCK occurs at the start of the first cycle of a data transfer.

### 9.2.5.4 enum \_ecspi\_interrupt\_enable

Enumerator

*kECSPI\_TxfifoEmptyInterruptEnable* Transmit FIFO buffer empty interrupt.

*kECSPI\_TxFifoDataRequestInterruptEnable* Transmit FIFO data request interrupt.

*kECSPI\_TxFifoFullInterruptEnable* Transmit FIFO full interrupt.

*kECSPI\_RxFifoReadyInterruptEnable* Receiver FIFO ready interrupt.

*kECSPI\_RxFifoDataRequestInterruptEnable* Receiver FIFO data request interrupt.

*kECSPI\_RxFifoFullInterruptEnable* Receiver FIFO full interrupt.

*kECSPI\_RxFifoOverflowInterruptEnable* Receiver FIFO buffer overflow interrupt.

*kECSPI\_TransferCompleteInterruptEnable* Transfer complete interrupt.

*kECSPI\_AllInterruptEnable* All interrupt.

### 9.2.5.5 enum \_ecspi\_flags

Enumerator

*kECSPI\_Tx fifoEmptyFlag* Transmit FIFO buffer empty flag.

*kECSPI\_TxFifoDataRequestFlag* Transmit FIFO data request flag.

*kECSPI\_TxFifoFullFlag* Transmit FIFO full flag.

*kECSPI\_RxFifoReadyFlag* Receiver FIFO ready flag.

*kECSPI\_RxFifoDataRequestFlag* Receiver FIFO data request flag.

*kECSPI\_RxFifoFullFlag* Receiver FIFO full flag.

*kECSPI\_RxFifoOverflowFlag* Receiver FIFO buffer overflow flag.

*kECSPI\_TransferCompleteFlag* Transfer complete flag.

## ECSPI Driver

### 9.2.5.6 enum \_ecspi\_dma\_enable\_t

Enumerator

*kECSPI\_TxDmaEnable* Tx DMA request source.

*kECSPI\_RxDMAEnable* Rx DMA request source.

*kECSPI\_DmaAllEnable* All DMA request source.

### 9.2.5.7 enum ecspi\_Data\_ready\_t

Enumerator

*kECSPI\_DataReadyIgnore* SPI\_RDY signal is ignored.

*kECSPI\_DataReadyFallingEdge* SPI\_RDY signal will be triggered by the falling edge.

*kECSPI\_DataReadyLowLevel* SPI\_RDY signal will be triggered by a low level.

### 9.2.5.8 enum ecspi\_channel\_source\_t

Enumerator

*kECSPI\_Channel0* Channel 0 is selected.

*kECSPI\_Channel1* Channel 1 is selected.

*kECSPI\_Channel2* Channel 2 is selected.

*kECSPI\_Channel3* Channel 3 is selected.

### 9.2.5.9 enum ecspi\_master\_slave\_mode\_t

Enumerator

*kECSPI\_Slave* ECSPI peripheral operates in slave mode.

*kECSPI\_Master* ECSPI peripheral operates in master mode.

### 9.2.5.10 enum ecspi\_data\_line\_inactive\_state\_t

Enumerator

*kECSPI\_DataLineInactiveStateHigh* The data line inactive state stays high.

*kECSPI\_DataLineInactiveStateLow* The data line inactive state stays low.

### 9.2.5.11 enum ecspi\_clock\_inactive\_state\_t

Enumerator

- kECSPI\_ClockInactiveStateLow* The SCLK inactive state stays low.
- kECSPI\_ClockInactiveStateHigh* The SCLK inactive state stays high.

### 9.2.5.12 enum ecspi\_chip\_select\_active\_state\_t

Enumerator

- kECSPI\_ChipSelectActiveStateLow* The SS signal line active stays low.
- kECSPI\_ChipSelectActiveStateHigh* The SS signal line active stays high.

### 9.2.5.13 enum ecspi\_wave\_form\_t

Enumerator

- kECSPI\_WaveFormSingle* The wave form for signal burst.
- kECSPI\_WaveFormMultiple* The wave form for multiple burst.

### 9.2.5.14 enum ecspi\_sample\_period\_clock\_source\_t

Enumerator

- kECSPI\_spiClock* The sample period clock source is SCLK.
- kECSPI\_lowFreqClock* The sample period clock source is low\_frequency reference clock(32.768 kHz).

## 9.2.6 Function Documentation

### 9.2.6.1 void ECSPI\_MasterGetDefaultConfig ( `ecspi_master_config_t * config` )

The purpose of this API is to get the configuration structure initialized for use in `ECSPI_MasterInit()`. User may use the initialized structure unchanged in `ECSPI_MasterInit`, or modify some fields of the structure before calling `ECSPI_MasterInit`. After calling this API, the master is ready to transfer. Example:

```
ecspi_master_config_t config;
ECSPI_MasterGetDefaultConfig(&config);
```

## ECSPI Driver

Parameters

<i>config</i>	pointer to config structure
---------------	-----------------------------

### 9.2.6.2 void ECSPI\_MasterInit ( **ECSPI\_Type** \* *base*, const **ecspi\_master\_config\_t** \* *config*, uint32\_t *srcClock\_Hz* )

The configuration structure can be filled by user from scratch, or be set with default values by [ECSPI-MasterGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
ecspi_master_config_t config = {  
    .baudRate_Bps = 400000,  
    ...  
};  
ECSPI_MasterInit(ECSPI0, &config);
```

Parameters

<i>base</i>	ECSPI base pointer
<i>config</i>	pointer to master configuration structure
<i>srcClock_Hz</i>	Source clock frequency.

### 9.2.6.3 void ECSPI\_SlaveGetDefaultConfig ( **ecspi\_slave\_config\_t** \* *config* )

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

```
ecspi_Slaveconfig_t config;  
ECSPI_SlaveGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to config structure
---------------	-----------------------------

### 9.2.6.4 void ECSPI\_SlaveInit ( **ECSPI\_Type** \* *base*, const **ecspi\_slave\_config\_t** \* *config* )

The configuration structure can be filled by user from scratch, or be set with default values by [ECSPI-SlaveGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
ecspi_Slaveconfig_t config = {  
    .baudRate_Bps = 400000,  
    ...  
};  
ECSPI_SlaveInit(ECSPI1, &config);
```

Parameters

<i>base</i>	ECSPI base pointer
<i>config</i>	pointer to master configuration structure

#### 9.2.6.5 void ECSPI\_Deinit ( **ECSPI\_Type** \* *base* )

Calling this API resets the ECSPI module, gates the ECSPI clock. The ECSPI module can't work unless calling the ECSPI\_MasterInit/ECSPI\_SlaveInit to initialize module.

Parameters

<i>base</i>	ECSPI base pointer
-------------	--------------------

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

Parameters

<i>base</i>	ECSPI base pointer
<i>enable</i>	pass true to enable module, false to disable module

#### 9.2.6.7 static uint32\_t ECSPI\_GetStatusFlags ( **ECSPI\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	ECSPI base pointer
-------------	--------------------

Returns

ECSPI Status, use status flag to AND [\\_ecspi\\_flags](#) could get the related status.

#### 9.2.6.8 static void ECSPI\_ClearStatusFlags ( **ECSPI\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

## ECSPI Driver

Parameters

<i>base</i>	ECSPI base pointer
<i>mask</i>	ECSPI Status, use status flag to AND <a href="#"><u>_ecspi_flags</u></a> could get the related status.

**9.2.6.9 static void ECSPI\_EnableInterrupts ( **ECSPI\_Type** \* *base*, **uint32\_t** *mask* )  
[inline], [static]**

Parameters

<i>base</i>	ECSPI base pointer
<i>mask</i>	ECSPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• kECSPI_Tx_fifoEmptyInterruptEnable</li><li>• kECSPI_Tx_FifoDataRequrstInterruptEnable</li><li>• kECSPI_Tx_FifoFullInterruptEnable</li><li>• kECSPI_Rx_FifoReadyInterruptEnable</li><li>• kECSPI_Rx_FifoDataRequrstInterruptEnable</li><li>• kECSPI_Rx_FifoFullInterruptEnable</li><li>• kECSPI_Rx_FifoOverFlowInterruptEnable</li><li>• kECSPI_TransferCompleteInterruptEnable</li><li>• kECSPI_AllInterruptEnable</li></ul>

**9.2.6.10 static void ECSPI\_DisableInterrupts ( *ECSPI\_Type* \* *base*, *uint32\_t* *mask* )  
[inline], [static]**

Parameters

<i>base</i>	ECSPI base pointer
<i>mask</i>	<p>ECSPI interrupt source. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> <li>• kECSPI_Tx_fifoEmptyInterruptEnable</li> <li>• kECSPI_Tx_FifoDataRequestInterruptEnable</li> <li>• kECSPI_Tx_FifoFullInterruptEnable</li> <li>• kECSPI_Rx_FifoReadyInterruptEnable</li> <li>• kECSPI_Rx_FifoDataRequestInterruptEnable</li> <li>• kECSPI_Rx_FifoFullInterruptEnable</li> <li>• kECSPI_Rx_FifoOverFlowInterruptEnable</li> <li>• kECSPI_TransferCompleteInterruptEnable</li> <li>• kECSPI_AllInterruptEnable</li> </ul>

**9.2.6.11 static void ECSPI\_SoftwareReset ( *ECSPI\_Type* \* *base* ) [inline], [static]**

Parameters

<i>base</i>	ECSPI base pointer
-------------	--------------------

**9.2.6.12 static bool ECSPI\_IsMaster ( *EC SPI\_Type* \* *base*, *ecspi\_channel\_source\_t* *channel* ) [inline], [static]**

Parameters

<i>base</i>	ECSPI base pointer
<i>channel</i>	ECSPI channel source

Returns

mode of channel

**9.2.6.13 static void ECSPI\_EnableDMA ( *EC SPI\_Type* \* *base*, *uint32\_t* *mask*, *bool* *enable* ) [inline], [static]**

## ECSPI Driver

Parameters

<i>base</i>	ECSPI base pointer
<i>source</i>	ECSPI DMA source.
<i>enable</i>	True means enable DMA, false means disable DMA

**9.2.6.14 static uint8\_t ECSPI\_GetTxFifoCount ( **ECSPI\_Type** \* *base* ) [inline], [static]**

Parameters

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

Returns

the number of words in Tx FIFO buffer.

**9.2.6.15 static uint8\_t ECSPI\_GetRxFifoCount ( **ECSPI\_Type** \* *base* ) [inline], [static]**

Parameters

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

Returns

the number of words in Rx FIFO buffer.

**9.2.6.16 static void ECSPI\_SetChannelSelect ( **ECSPI\_Type** \* *base*, **ecspi\_channel\_source\_t** *channel* ) [inline], [static]**

Parameters

<i>base</i>	ECSPI base pointer
-------------	--------------------

<i>channel</i>	Channel source.
----------------	-----------------

### 9.2.6.17 void ECSPI\_SetChannelConfig ( **ECSPI\_Type** \* *base*, **ecspi\_channel\_source\_t** *channel*, **const ecspi\_channel\_config\_t** \* *config* )

The purpose of this API is to set the channel will be use to transfer. User may use this API after instance has been initialized or before transfer start. The configuration structure `#_ecspi_channel_config_` can be filled by user from scratch. After calling this API, user can select this channel as transfer channel.

Parameters

<i>base</i>	ECSPI base pointer
<i>channel</i>	Channel source.
<i>config</i>	Configuration struct of channel

### 9.2.6.18 void ECSPI\_SetBaudRate ( **ECSPI\_Type** \* *base*, **uint32\_t** *baudRate\_Bps*, **uint32\_t** *srcClock\_Hz* )

This is only used in master.

Parameters

<i>base</i>	ECSPI base pointer
<i>baudRate_Bps</i>	baud rate needed in Hz.
<i>srcClock_Hz</i>	ECSPI source clock frequency in Hz.

### 9.2.6.19 void ECSPI\_WriteBlocking ( **ECSPI\_Type** \* *base*, **uint32\_t** \* *buffer*, **size\_t** *size* )

Note

This function blocks via polling until all bytes have been sent.

Parameters

<i>base</i>	ECSPI base pointer
-------------	--------------------

## ECSPI Driver

<i>buffer</i>	The data bytes to send
<i>size</i>	The number of data bytes to send

### 9.2.6.20 void ECSPI\_WriteData ( **ECSPI\_Type** \* *base*, **uint32\_t** *data* ) [inline]

Parameters

<i>base</i>	ECSPI base pointer
<i>data</i>	Data needs to be write.

### 9.2.6.21 **uint32\_t** ECSPI\_ReadData ( **ECSPI\_Type** \* *base* ) [inline]

Parameters

<i>base</i>	ECSPI base pointer
-------------	--------------------

Returns

Data in the register.

### 9.2.6.22 void ECSPI\_MasterTransferCreateHandle ( **ECSPI\_Type** \* *base*, **ecspi\_master\_handle\_t** \* *handle*, **ecspi\_master\_callback\_t** *callback*, **void** \* *userData* )

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

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	ECSPI handle pointer.
<i>callback</i>	Callback function.
<i>userData</i>	User data.

### 9.2.6.23 **status\_t** ECSPI\_MasterTransferBlocking ( **ECSPI\_Type** \* *base*, **ecspi\_transfer\_t** \* *xfer* )

Parameters

<i>base</i>	SPI base pointer
<i>xfer</i>	pointer to <code>spi_xfer_config_t</code> structure

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.

#### 9.2.6.24 `status_t ECSPI_MasterTransferNonBlocking ( ECSPI_Type * base, ecspi_master_handle_t * handle, ecspi_transfer_t * xfer )`

Note

The API immediately returns after transfer initialization is finished.

If ECSPI transfer data frame size is 16 bits, the transfer size cannot be an odd number.

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	pointer to <code>ecspi_master_handle_t</code> structure which stores the transfer state
<i>xfer</i>	pointer to <code>ecspi_transfer_t</code> structure

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_ECSPI_Busy</i>	ECSPI is not idle, is running another transfer.

#### 9.2.6.25 `status_t ECSPI_MasterTransferGetCount ( ECSPI_Type * base, ecspi_master_handle_t * handle, size_t * count )`

Parameters

## ECSPI Driver

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	Pointer to ECSPI transfer handle, this should be a static variable.
<i>count</i>	Transferred bytes of ECSPI master.

Return values

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

**9.2.6.26 void ECSPI\_MasterTransferAbort ( *ECSPI\_Type* \* *base*, *ecspi\_master\_handle\_t* \* *handle* )**

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	Pointer to ECSPI transfer handle, this should be a static variable.

**9.2.6.27 void ECSPI\_MasterTransferHandleIRQ ( *ECSPI\_Type* \* *base*, *ecspi\_master\_handle\_t* \* *handle* )**

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	pointer to <i>ecspi_master_handle_t</i> structure which stores the transfer state.

**9.2.6.28 void ECSPI\_SlaveTransferCreateHandle ( *ECSPI\_Type* \* *base*, *ecspi\_slave\_handle\_t* \* *handle*, *ecspi\_slave\_callback\_t* *callback*, *void* \* *userData* )**

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

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	ECSPI handle pointer.
<i>callback</i>	Callback function.
<i>userData</i>	User data.

### 9.2.6.29 static status\_t ECSPISlaveTransferNonBlocking ( **ECSPI\_Type** \* *base*, **ecspi\_slave\_handle\_t** \* *handle*, **ecspi\_transfer\_t** \* *xfer* ) [inline], [static]

Note

The API returns immediately after the transfer initialization is finished.

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	pointer to <b>ecspi_master_handle_t</b> structure which stores the transfer state
<i>xfer</i>	pointer to <b>ecspi_transfer_t</b> structure

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_ECSPI_Busy</i>	EC SPI is not idle, is running another transfer.

### 9.2.6.30 static status\_t ECSPISlaveTransferGetCount ( **ECSPI\_Type** \* *base*, **ecspi\_slave\_handle\_t** \* *handle*, **size\_t** \* *count* ) [inline], [static]

Parameters

<i>base</i>	EC SPI peripheral base address.
<i>handle</i>	Pointer to EC SPI transfer handle, this should be a static variable.
<i>count</i>	Transferred bytes of EC SPI slave.

Return values

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

9.2.6.31 **static void ECSPI\_SlaveTransferAbort ( ECSPI\_Type \* *base*,  
ecspi\_slave\_handle\_t \* *handle* ) [inline], [static]**

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	Pointer to ECSPI transfer handle, this should be a static variable.

#### 9.2.6.32 void ECSPI\_SlaveTransferHandleIRQ ( **ECSPI\_Type** \* *base*, **ecspi\_slave\_handle\_t** \* *handle* )

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	pointer to <b>ecspi_slave_handle_t</b> structure which stores the transfer state

### 9.3 ECSPI FreeRTOS Driver

#### 9.3.1 Overview

#### ECSPI RTOS Operation

- status\_t **ECSPI\_RTOS\_Init** (ecspi\_rtos\_handle\_t \*handle, ECSPI\_Type \*base, const **ecspi\_master\_config\_t** \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes ECSPI.*
- status\_t **ECSPI\_RTOS\_Deinit** (ecspi\_rtos\_handle\_t \*handle)  
*Deinitializes the ECSPI.*
- status\_t **ECSPI\_RTOS\_Transfer** (ecspi\_rtos\_handle\_t \*handle, **ecspi\_transfer\_t** \*transfer)  
*Performs ECSPI transfer.*

#### 9.3.2 Function Documentation

##### 9.3.2.1 status\_t **ECSPI\_RTOS\_Init** ( **ecspi\_rtos\_handle\_t** \* *handle*, **ECSPI\_Type** \* *base*, const **ecspi\_master\_config\_t** \* *masterConfig*, uint32\_t *srcClock\_Hz* )

This function initializes the ECSPI module and related RTOS context.

Parameters

<i>handle</i>	The RTOS ECSPI handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the ECSPI instance to initialize.
<i>masterConfig</i>	Configuration structure to set-up ECSPI in master mode.
<i>srcClock_Hz</i>	Frequency of input clock of the ECSPI module.

Returns

status of the operation.

##### 9.3.2.2 status\_t **ECSPI\_RTOS\_Deinit** ( **ecspi\_rtos\_handle\_t** \* *handle* )

This function deinitializes the ECSPI module and related RTOS context.

Parameters

<i>handle</i>	The RTOS ECSPI handle.
---------------	------------------------

### 9.3.2.3 **status\_t ECSPI\_RTOS\_Transfer ( *ecspi\_rtos\_handle\_t \* handle*, *ecspi\_transfer\_t \* transfer* )**

This function performs an ECSPI transfer according to data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS ECSPI handle.
<i>transfer</i>	Structure specifying the transfer parameters.

Returns

status of the operation.



# Chapter 10

## eLCDIF: Enhanced LCD Interface

### 10.1 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.

### 10.2 Typical use case

#### 10.2.1 Frame buffer update

The function ELCDIF\_SetNextBufferAddr sets the next frame to show to eLCDIF, the eLCDIF load the new frame and set the interrupt [kELCDIF\\_CurFrameDone](#). If no new frame set, the old one is displayed.

```
volatile bool s_frameDone = false;

void LCDIF_IRQHandler(void)
{
    uint32_t intStatus;

    intStatus = ELCDIF_GetInterruptStatus(TEST_ELCDIF);

    ELCDIF_ClearInterruptStatus(TEST_ELCDIF, intStatus);

    if (intStatus & kELCDIF_CurFrameDone)
    {
        s_frameDone = true;
    }
}

void main(void)
{
    // Initialize the eLCDIF. */
    const elcdif_rgb_mode_config_t config =
    {
        // ...
    };

    ELCDIF_RgbModeInit(LCDIF, &config);

    /*
     * Now the eLCDIF has not started to display, application could do other
     * initialize work here.
     */

    /* Enable interrupt and start to display. */
    ELCDIF_EnableInterrupts(LCDIF,
                           kELCDIF_CurFrameDoneInterruptEnable);
    ELCDIF_RgbModeStart(LCDIF);

    while (1)
    {
        /* Prepare the new frame here. */
```

## Typical use case

```
// ...

ELCDIF_SetNextBufferAddr(LCDIF, frameBuffer);

/* Wait the previous frame loaded to eLCDIF. */
while (!s_frameDone)
{
}
}
```

### 10.2.2 Alpha surface

The alpha surface could 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.

```
volatile bool s_frameDone = false;

void LCDIF_IRQHandler(void)
{
    uint32_t intStatus;

    intStatus = ELCDIF_GetInterruptStatus(TEST_ELCDIF);

    ELCDIF_ClearInterruptStatus(TEST_ELCDIF, intStatus);

    if (intStatus & kELCDIF_CurFrameDone)
    {
        s_frameDone = true;
    }
}

void main(void)
{
    /* Initialize the eLCDIF. */
    const elcdif_rgb_mode_config_t config =
    {
        // ...
    };

    ELCDIF_RgbModeInit(LCDIF, &config);

    /* Configure the alpha surface. */
    const elcdif_as_buffer_config_t asBufferConfig =
    {
        .bufferAddr = (uint32_t)s_asBuffer,
        .pixelFormat = kELCDIF_AsPixelFormatARGB8888,
    };

    elcdif_as_blend_config_t asBlendConfig =
    {
        .alpha = 0,
        .invertAlpha = false,
        .alphaMode = kELCDIF_AlphaOverride,
        .ropMode = kELCDIF_RopMaskAs, /* Don't care. */
    };

    ELCDIF_SetAlphaSurfaceBufferConfig(ELCDIF, &asBufferConfig);
    ELCDIF_SetAlphaSurfaceBlendConfig(LCDIF, &asBlendConfig);
    ELCDIF_EnableAlphaSurface(ELCDIF, true);

    /*
     * Now the eLCDIF has not started to display, application could do other

```

```

    * initialize work here.
    */

/* Enable interrupt and start to display. */
eLCDIF_EnableInterrupts(LCDIF,
    kELCDIF_CurFrameDoneInterruptEnable);
eLCDIF_RgbModeStart(LCDIF);

while (1)
{
    /* Prepare the new frame here. */
    // ...

    /* Wait the previous frame loaded to eLCDIF. */
    while (!s_frameDone)
    {
    }

    /* Change to use the new alpha value. */
    asBlendConfig.alpha = newAlphaValue;
}
}

```

## 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...](#)

## 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,
 kELCDIF\_SciSyncOnInterruptEnable }
 *The eLCDIF interrupts to enable.*

## Typical use case

- 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,  
    `kELCDIF_SciSyncOn` = LCDIF\_AS\_CTRL\_CSI\_SYNC\_ON\_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,  
    `kELCDIF_LcdControllerBusy` = LCDIF\_STAT\_BUSY\_MASK,  
    `kELCDIF_CurDviField2` = LCDIF\_STAT\_DVI\_CURRENT\_FIELD\_MASK }
- The *eLCDIF status flags*.
- enum `elcdif_pixel_format_t` {  
    `kELCDIF_PixelFormatRAW8` = 0,  
    `kELCDIF_PixelFormatRGB565` = 1,  
    `kELCDIF_PixelFormatRGB666` = 2,  
    `kELCDIF_PixelFormatRGB888` = 3 }
- 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 }
- The *eLCDIF alpha surface pixel format*.
- enum `elcdif_alpha_mode_t` {  
    `kELCDIF_AlphaEmbedded`,  
    `kELCDIF_AlphaOverride`,  
    `kELCDIF_AlphaMultiply`,  
    `kELCDIF_AlphaRop` }
- The *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.*

## 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).*

## 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.*

## Alpha surface

- void **ELCDIF\_SetAlphaSurfaceBufferConfig** (LCDIF\_Type \*base, const **elcdif\_as\_buffer\_config\_t** \*config)  
*Set the configuration for alpha surface buffer.*
- void **ELCDIF\_SetAlphaSurfaceBlendConfig** (LCDIF\_Type \*base, const **elcdif\_as\_blend\_config\_t** \*config)  
*Set the alpha surface blending configuration.*
- static void **ELCDIF\_SetNextAlphaSurfaceBufferAddr** (LCDIF\_Type \*base, uint32\_t bufferAddr)  
*Set the next alpha surface buffer address.*

## Data Structure Documentation

- static void [ELCDIF\\_SetOverlayColorKey](#) (LCDIF\_Type \*base, uint32\_t colorKeyLow, uint32\_t colorKeyHigh)  
*Set the overlay color key.*
- static void [ELCDIF\\_EnableOverlayColorKey](#) (LCDIF\_Type \*base, bool enable)  
*Enable or disable the color key.*
- static void [ELCDIF\\_EnableAlphaSurface](#) (LCDIF\_Type \*base, bool enable)  
*Enable or disable the alpha surface.*
- static void [ELCDIF\\_EnableProcessSurface](#) (LCDIF\_Type \*base, bool enable)  
*Enable or disable the process surface.*

### 10.3 Data Structure Documentation

#### 10.3.1 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.*

##### 10.3.1.0.0.3 Field Documentation

###### 10.3.1.0.0.3.1 uint32\_t elcdif\_pixel\_format\_reg\_t::regCtrl

###### 10.3.1.0.0.3.2 uint32\_t elcdif\_pixel\_format\_reg\_t::regCtrl1

#### 10.3.2 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

- `uint8_t vsw`  
*Horizontal back porch.*
- `uint8_t vfp`  
*VSYNC pulse width.*
- `uint8_t vbp`  
*Vertical front porch.*
- `uint8_t polarityFlags`  
*Vertical back porch.*
- `uint32_t bufferAddr`  
*OR'ed value of `_elcdif_polarity_flags`, used to control the signal polarity.*
- `uint32_t pixelFormat`  
*Frame buffer address.*
- `elcdif_pixel_format_t pixelFormat`  
*Pixel format.*
- `elcdif_lcd_data_bus_t dataBus`  
*LCD data bus.*

#### 10.3.2.0.0.4 Field Documentation

**10.3.2.0.0.4.1 `uint16_t elcdif_rgb_mode_config_t::panelWidth`**

**10.3.2.0.0.4.2 `uint16_t elcdif_rgb_mode_config_t::panelHeight`**

**10.3.2.0.0.4.3 `uint8_t elcdif_rgb_mode_config_t::hsw`**

**10.3.2.0.0.4.4 `uint8_t elcdif_rgb_mode_config_t::hfp`**

**10.3.2.0.0.4.5 `uint8_t elcdif_rgb_mode_config_t::hbp`**

**10.3.2.0.0.4.6 `uint8_t elcdif_rgb_mode_config_t::vsw`**

**10.3.2.0.0.4.7 `uint8_t elcdif_rgb_mode_config_t::vfp`**

**10.3.2.0.0.4.8 `uint8_t elcdif_rgb_mode_config_t::vbp`**

**10.3.2.0.0.4.9 `uint32_t elcdif_rgb_mode_config_t::polarityFlags`**

**10.3.2.0.0.4.10 `uint32_t elcdif_rgb_mode_config_t::bufferAddr`**

**10.3.2.0.0.4.11 `elcdif_pixel_format_t elcdif_rgb_mode_config_t::pixelFormat`**

**10.3.2.0.0.4.12 `elcdif_lcd_data_bus_t elcdif_rgb_mode_config_t::dataBus`**

### 10.3.3 `struct elcdif_as_buffer_config_t`

#### Data Fields

- `uint32_t bufferAddr`  
*Buffer address.*
- `elcdif_as_pixel_format_t pixelFormat`  
*Pixel format.*

## Enumeration Type Documentation

### 10.3.3.0.0.5 Field Documentation

10.3.3.0.0.5.1 `uint32_t elcdif_as_buffer_config_t::bufferAddr`

10.3.3.0.0.5.2 `elcdif_as_pixel_format_t elcdif_as_buffer_config_t::pixelFormat`

### 10.3.4 `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`.*

### 10.3.4.0.0.6 Field Documentation

10.3.4.0.0.6.1 `uint8_t elcdif_as_blend_config_t::alpha`

10.3.4.0.0.6.2 `bool elcdif_as_blend_config_t::invertAlpha`

10.3.4.0.0.6.3 `elcdif_alpha_mode_t elcdif_as_blend_config_t::alphaMode`

10.3.4.0.0.6.4 `elcdif_rop_mode_t elcdif_as_blend_config_t::ropMode`

## 10.4 Macro Definition Documentation

10.4.1 `#define FSL_ELCDIF_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

Version 2.0.0.

## 10.5 Enumeration Type Documentation

### 10.5.1 `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.

### 10.5.2 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.

***kELCDIF\_SciSyncOnInterruptEnable*** Interrupt when eLCDIF lock with CSI input.

### 10.5.3 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.

***kELCDIF\_SciSyncOn*** Interrupt when eLCDIF lock with CSI input.

### 10.5.4 enum \_elcdif\_status\_flags

Enumerator

***kELCDIF\_LFifoFull*** LFIFO full.

***kELCDIF\_LFifoEmpty*** LFIFO empty.

***kELCDIF\_TxFifoFull*** TXFIFO full.

***kELCDIF\_TxFifoEmpty*** TXFIFO empty.

***kELCDIF\_LcdControllerBusy*** The external LCD controller busy signal.

***kELCDIF\_CurDviField2*** Current DVI filed, if set, then current filed is 2, otherwise current filed is 1.

## Enumeration Type Documentation

### 10.5.5 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.

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\_PixelFormatRGB888* RGB888 unpacked, one pixel uses 32 bits, high byte unused.

### 10.5.6 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.

### 10.5.7 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.

### 10.5.8 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 to 200, user defined alpha set to 100, then the result alpha is  $200 * 100 / 255$ .

*kLCDIF\_AlphaRop* Raster operation.

### 10.5.9 enum elcdif\_rop\_mode\_t

Explanation:

- AS: Alpha surface
- PS: Process surface
- nAS: Alpha surface NOT value
- nPS: Process surface NOT value

Enumerator

```

kLCDIF_RopMaskAs AS AND PS.
kLCDIF_RopMaskNotAs nAS AND PS.
kLCDIF_RopMaskAsNot AS AND nPS.
kLCDIF_RopMergeAs AS OR PS.
kLCDIF_RopMergeNotAs nAS OR PS.
kLCDIF_RopMergeAsNot AS OR nPS.
kLCDIF_RopNotCopyAs nAS.
kLCDIF_RopNot nPS.
kLCDIF_RopNotMaskAs AS NAND PS.
kLCDIF_RopNotMergeAs AS NOR PS.
kLCDIF_RopXorAs AS XOR PS.
kLCDIF_RopNotXorAs AS XNOR PS.

```

## 10.6 Function Documentation

### 10.6.1 void LCDIF\_RgbModelInit ( 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.

Parameters

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

### 10.6.2 static uint32\_t LCDIF\_GetStatus ( LCDIF\_Type \* *base* ) [inline], [static]

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

## Function Documentation

```
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);

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;
    }
}

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](#).

### 10.6.3 static uint32\_t ELCDIF\_GetLFifoCount ( LCDIF\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

The LFIFO current count

### 10.6.4 static void ELCDIF\_EnableInterrupts ( 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 <a href="#">_elcdif_interrupt_enable</a> .

### 10.6.5 static void ELCDIF\_DisableInterrupts ( 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 _elcdif_interrupt_enable.

**10.6.6 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.

**10.6.7 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 _elcdif_interrupt_flags.

**10.6.8 void ELCDIF\_SetAlphaSurfaceBufferConfig ( LCDIF\_Type \* *base*, const  
elcdif\_as\_buffer\_config\_t \* *config* )**

Parameters

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

**10.6.9 void ELCDIF\_SetAlphaSurfaceBlendConfig ( LCDIF\_Type \* *base*, const  
elcdif\_as\_blend\_config\_t \* *config* )**

Parameters

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

#### 10.6.10 static void ELCDIF\_SetNextAlphaSurfaceBufferAddr ( LCDIF\_Type \* *base*, uint32\_t *bufferAddr* ) [inline], [static]

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>bufferAddr</i>	Alpha surface buffer address.

#### 10.6.11 static void ELCDIF\_SetOverlayColorKey ( LCDIF\_Type \* *base*, uint32\_t *colorKeyLow*, uint32\_t *colorKeyHigh* ) [inline], [static]

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.

Parameters

<i>base</i>	eLCDIF 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

#### 10.6.12 static void ELCDIF\_EnableOverlayColorKey ( LCDIF\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

## Function Documentation

<i>base</i>	eLCDIF peripheral base address.
<i>enable</i>	True to enable, false to disable.

**10.6.13 static void ELCDIF\_EnableAlphaSurface ( LCDIF\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>enable</i>	True to enable, false to disable.

**10.6.14 static void ELCDIF\_EnableProcessSurface ( LCDIF\_Type \* *base*, bool *enable* ) [inline], [static]**

Process surface is the normal frame buffer. The process surface content is controlled by ELCDIF\_SetNextBufferAddr.

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>enable</i>	True to enable, false to disable.

# Chapter 11

## ENET: Ethernet MAC Driver

### 11.1 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 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\\_AddMulticast-Group\(\)](#) should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

For ENET receive, the [ENET\\_GetRxFrameSize\(\)](#) function must 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.

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 must 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.

### 11.2 Typical use case

#### 11.2.1 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.

## Typical use case

```
enet_config_t config;
uint32_t length = 0;
uint32_t sysClock;
uint32_t phyAddr = 0;
bool link = false;
phy_speed_t speed;
phy_duplex_t duplex;
enet_status_t result;
enet_data_error_stats_t eErrorStatic;
// Prepares the buffer configuration.
enet_buffer_config_t buffCfg =
{
    ENET_RXBD_NUM,
    ENET_TXBD_NUM,
    ENET_BuffSizeAlign(ENET_RXBUFF_SIZE),
    ENET_BuffSizeAlign(ENET_TXBUFF_SIZE),
    &RxBuffDesrip[0], // Prepare buffers
    &TxBuffDesrip[0], // Prepare buffers
    &RxDataBuff[0][0], // Prepare buffers
    &TxDataBuff[0][0], // Prepare buffers
};

sysClock = CLOCK_GetFreq(kCLOCK_CoreSysClk);

// Gets the default configuration.
ENET_GetDefaultConfig(&config);
PHY_Init(EXAMPLE_ENET, 0, sysClock);
// Changes the link status to PHY auto-negotiated link status.
PHY_GetLinkStatus(EXAMPLE_ENET, phyAddr, &link);
if (link)
{
    PHY_GetLinkSpeedDuplex(EXAMPLE_ENET, phyAddr, &speed, &duplex);
    config.miSpeed = (enet_mi_speed_t)speed;
    config.miDuplex = (enet_mi_duplex_t)duplex;
}
ENET_Init(EXAMPLE_ENET, &handle, &config, &buffCfg, &macAddr[0], sysClock);
ENET_ActiveRead(EXAMPLE_ENET);

while (1)
{
    // Gets the frame size.
    result = ENET_GetRxFrameSize(&handle, &length);
    // Calls the ENET_ReadFrame when there is a received frame.
    if (length != 0)
    {
        // Receives a valid frame and delivers the receive buffer with the size equal to length.
        uint8_t *data = (uint8_t *)malloc(length);
        ENET_ReadFrame(EXAMPLE_ENET, &handle, data, length);
        // Delivers the data to the upper layer.
        .....
        free(data);
    }
    else if (result == kStatus_ENET_RxFrameErr)
    {
        // Updates the received buffer when an error occurs.
        ENET_GetRxErrBeforeReadFrame(&handle, &eErrStatic);
        // Updates the receive buffer.
        ENET_ReadFrame(EXAMPLE_ENET, &handle, NULL, 0);
    }

    // Sends a multicast frame when the PHY is linked up.
    if (kStatus_Success == PHY_GetLinkStatus(EXAMPLE_ENET, phyAddr, &link))
    {
        if (link)
        {
            ENET_SendFrame(EXAMPLE_ENET, &handle, &frame[0], ENET_DATA_LENGTH);
        }
    }
}
```

}

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.

```

enet_config_t config;
uint32_t length = 0;
uint32_t sysClock;
uint32_t phyAddr = 0;
bool link = false;
phy_speed_t speed;
phy_duplex_t duplex;
enet_status_t result;
enet_data_err_stats_t eErrStatic;
enet_buffer_config_t buffCfg =
{
    ENET_RXBD_NUM,
    ENET_TXBD_NUM,
    ENET_BuffSizeAlign(ENET_RXBUFF_SIZE),
    ENET_BuffSizeAlign(ENET_TXBUFF_SIZE),
    &RxBuffDesrip[0],
    &TxBuffDesrip[0],
    &RxDataBuff[0][0],
    &TxDataBuff[0][0],
};

sysClock = CLOCK_GetFreq(kCLOCK_CoreSysClk);

// Sets the PTP 1588 source.
CLOCK_SetEnetTime0Clock(2);
ptpClock = CLOCK_GetFreq(kCLOCK_Osc0ErClk);
// Prepares the PTP configuration.
enet_ptp_config_t ptpConfig =
{
    ENET_RXBD_NUM,
    ENET_TXBD_NUM,
    &g_rxPtpTsBuff[0],
    &g_txPtpTsBuff[0],
    kENET_PtpTimerChannel1,
    ptpClock,
};

// Gets the default configuration.
ENET_GetDefaultConfig(&config);

PHY_Init(EXAMPLE_ENET, 0, sysClock);
// Changes the link status to PHY auto-negotiated link status.
PHY_GetLinkStatus(EXAMPLE_ENET, phyAddr, &link);
if (link)
{
    PHY_GetLinkSpeedDuplex(EXAMPLE_ENET, phyAddr, &speed, &duplex);
    config.miiSpeed = (enet_mii_speed_t)speed;
    config.miiDuplex = (enet_mii_duplex_t)duplex;
}

ENET_Init(EXAMPLE_ENET, &handle, &config, &buffCfg, &macAddr[0], sysClock);

// Configures the PTP 1588 feature.
ENET_Ptp1588Configure(EXAMPLE_ENET, &handle, &ptpConfig);
// Adds the device to the PTP multicast group.
ENET_AddMulticastGroup(EXAMPLE_ENET, &mGAddr[0]);

ENET_ActiveRead(EXAMPLE_ENET);

```

## Typical use case

## 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_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.*
- `#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)`  
*Defines the PHY address scope for the ENET.*

## 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_TxFrameBusy`,  
  `kStatus_ENET_TxFrameFail` = MAKE\_STATUS(kStatusGroup\_ENET, 4U) }  
*Defines the status return codes for transaction.*
- enum `enet_mii_mode_t` {  
  `kENET_MiiMode` = 0U,  
  `kENET_RmiiMode` }  
*Defines the RMII or MII mode for data interface between the MAC and the PHY.*

- enum `enet_mii_speed_t` {
   
  kENET\_MiiSpeed10M = 0U,
   
  kENET\_MiiSpeed100M }
   
*Defines the 10 Mbps or 100 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 }
   
*Defines the write operation for the MII management frame.*
- enum `enet_mii_read_t` {
   
  kENET\_MiiReadValidFrame = 2U,
   
  kENET\_MiiReadNoCompliant = 3U }
   
*Defines the read operation for the MII management frame.*
- enum `enet_mii_extend_opcode` {
   
  kENET\_MiiAddrWrite\_C45 = 0U,
   
  kENET\_MiiWriteFrame\_C45 = 1U,
   
  kENET\_MiiReadFrame\_C45 = 3U }
   
*Define the MII opcode for extended MDIO\_CLAUSES\_45 Frame.*
- enum `enet_special_control_flag_t` {
   
  kENET\_ControlFlowControlEnable = 0x0001U,
   
  kENET\_ControlRxPayloadCheckEnable = 0x0002U,
   
  kENET\_ControlRxPadRemoveEnable = 0x0004U,
   
  kENET\_ControlRxBroadCastRejectEnable = 0x0008U,
   
  kENET\_ControlMacAddrInsert = 0x0010U,
   
  kENET\_ControlStoreAndFwdDisable = 0x0020U,
   
  kENET\_ControlSMIPreambleDisable = 0x0040U,
   
  kENET\_ControlPromiscuousEnable = 0x0080U,
   
  kENET\_ControlMIILoopEnable = 0x0100U,
   
  kENET\_ControlVLANTagEnable = 0x0200U }
   
*Defines a special configuration for ENET MAC controller.*
- enum `enet_interrupt_enable_t` {

## Typical use case

```
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.*

## Driver version

- #define `FSL_ENET_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 1))  
*Defines the driver version.*

## Control and status region bit masks of the receive buffer descriptor.

- #define `ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK` 0x8000U  
*Empty bit mask.*

- #define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER1\_MASK 0x4000U  
*Software owner one mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_WRAP\_MASK 0x2000U  
*Next buffer descriptor is the start address.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER2\_MASK 0x1000U  
*Software owner two mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_LAST\_MASK 0x0800U  
*Last BD of the frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_MISS\_MASK 0x0100U  
*Received because of the promiscuous mode.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_BROADCAST\_MASK 0x0080U  
*Broadcast packet mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_MULTICAST\_MASK 0x0040U  
*Multicast packet mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_LENVLIOLATE\_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\_TRANMICRC\_MASK 0x0400U  
*Transmit CRC mask.*

## Defines the maximum Ethernet frame size.

- #define ENET\_FRAME\_MAX\_FRAMELEN 1518U  
*Default maximum Ethernet frame size.*

## 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)

## Typical use case

- `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 operations

- `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 the 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 \*eErrorStatic)  
*Gets the ENET the error statistics of a received frame.*
- status\_t **ENET\_GetRxFrameSize** (enet\_handle\_t \*handle, uint32\_t \*length)  
*Gets the size of the read frame.*
- status\_t **ENET\_ReadFrame** (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t \*data, uint32\_t length)  
*Reads a frame from the ENET device.*
- status\_t **ENET\_SendFrame** (ENET\_Type \*base, enet\_handle\_t \*handle, const uint8\_t \*data, uint32\_t length)  
*Transmits an ENET frame.*
- 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)  
*The error IRQ handler.*
- void **ENET\_CommonFrame0IRQHandler** (ENET\_Type \*base)  
*the common IRQ handler for the tx/rx/error etc irq handler.*

## 11.3 Data Structure Documentation

### 11.3.1 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.*

## Data Structure Documentation

### 11.3.1.0.0.7 Field Documentation

11.3.1.0.0.7.1 `uint16_t enet_rx_bd_struct_t::length`

11.3.1.0.0.7.2 `uint16_t enet_rx_bd_struct_t::control`

11.3.1.0.0.7.3 `uint8_t* enet_rx_bd_struct_t::buffer`

### 11.3.2 `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.*

### 11.3.2.0.0.8 Field Documentation

11.3.2.0.0.8.1 `uint16_t enet_tx_bd_struct_t::length`

11.3.2.0.0.8.2 `uint16_t enet_tx_bd_struct_t::control`

11.3.2.0.0.8.3 `uint8_t* enet_tx_bd_struct_t::buffer`

### 11.3.3 `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.*

### 11.3.3.0.0.9 Field Documentation

**11.3.3.0.0.9.1 uint32\_t enet\_data\_error\_stats\_t::statsRxLenGreaterErr**

**11.3.3.0.0.9.2 uint32\_t enet\_data\_error\_stats\_t::statsRxFcsErr**

**11.3.3.0.0.9.3 uint32\_t enet\_data\_error\_stats\_t::statsRxOverRunErr**

**11.3.3.0.0.9.4 uint32\_t enet\_data\_error\_stats\_t::statsRxTruncateErr**

### 11.3.4 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\_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.*
- **volatile enet\_tx\_bd\_struct\_t \* txBdStartAddrAlign**  
*Aligned transmit buffer descriptor start address.*
- **uint8\_t \* rxBufferAlign**  
*Receive data buffer start address.*
- **uint8\_t \* txBufferAlign**  
*Transmit data buffer start address.*

## Data Structure Documentation

### 11.3.4.0.0.10 Field Documentation

- 11.3.4.0.0.10.1 `uint16_t enet_buffer_config_t::rxBdNumber`
- 11.3.4.0.0.10.2 `uint16_t enet_buffer_config_t::txBdNumber`
- 11.3.4.0.0.10.3 `uint32_t enet_buffer_config_t::rxBuffSizeAlign`
- 11.3.4.0.0.10.4 `uint32_t enet_buffer_config_t::txBuffSizeAlign`
- 11.3.4.0.0.10.5 `volatile enet_rx_bd_struct_t* enet_buffer_config_t::rxBdStartAddrAlign`
- 11.3.4.0.0.10.6 `volatile enet_tx_bd_struct_t* enet_buffer_config_t::txBdStartAddrAlign`
- 11.3.4.0.0.10.7 `uint8_t* enet_buffer_config_t::rxBufferAlign`
- 11.3.4.0.0.10.8 `uint8_t* enet_buffer_config_t::txBufferAlign`

### 11.3.5 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.*

### 11.3.5.0.0.11 Field Documentation

- 11.3.5.0.0.11.1 `uint8_t enet_intcoalesce_config_t::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- 11.3.5.0.0.11.2 `uint16_t enet_intcoalesce_config_t::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`
- 11.3.5.0.0.11.3 `uint8_t enet_intcoalesce_config_t::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- 11.3.5.0.0.11.4 `uint16_t enet_intcoalesce_config_t::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

### 11.3.6 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 if 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.

## Data Fields

- **uint32\_t macSpecialConfig**  
*Mac special configuration.*
- **uint32\_t interrupt**  
*Mac interrupt source.*
- **uint16\_t rxMaxFrameLen**  
*Receive maximum frame length.*
- **enet\_mii\_mode\_t miiMode**  
*MII mode.*
- **enet\_mii\_speed\_t miiSpeed**  
*MII Speed.*
- **enet\_mii\_duplex\_t miiDuplex**  
*MII duplex.*
- **uint8\_t rxAccelerConfig**  
*Receive accelerator, A logical OR of "enet\_rx\_accelerator\_t".*
- **uint8\_t txAccelerConfig**  
*Transmit accelerator, A logical OR of "enet\_rx\_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.*

## Data Structure Documentation

- `uint8_t txFifoWatermark`

For store and forward disable case, the data required in TX FIFO before a frame transmit start.

### 11.3.6.0.0.12 Field Documentation

#### 11.3.6.0.0.12.1 `uint32_t enet_config_t::macSpecialConfig`

A logical OR of "enet\_special\_control\_flag\_t".

#### 11.3.6.0.0.12.2 `uint32_t enet_config_t::interrupt`

A logical OR of "enet\_interrupt\_enable\_t".

#### 11.3.6.0.0.12.3 `uint16_t enet_config_t::rxMaxFrameLen`

#### 11.3.6.0.0.12.4 `enet_mii_mode_t enet_config_t::miiMode`

#### 11.3.6.0.0.12.5 `enet_mii_speed_t enet_config_t::miiSpeed`

#### 11.3.6.0.0.12.6 `enet_mii_duplex_t enet_config_t::miiDuplex`

#### 11.3.6.0.0.12.7 `uint8_t enet_config_t::rxAccelerConfig`

#### 11.3.6.0.0.12.8 `uint8_t enet_config_t::txAccelerConfig`

#### 11.3.6.0.0.12.9 `uint16_t enet_config_t::pauseDuration`

#### 11.3.6.0.0.12.10 `uint8_t enet_config_t::rx_fifoEmptyThreshold`

#### 11.3.6.0.0.12.11 `uint8_t enet_config_t::rx_fifoFullThreshold`

#### 11.3.6.0.0.12.12 `uint8_t enet_config_t::tx_fifoWatermark`

### 11.3.7 `struct _enet_handle`

## Data Fields

- volatile `enet_rx_bd_struct_t * rxBdBase`  
*Receive buffer descriptor base address pointer.*
- volatile `enet_rx_bd_struct_t * rxBdCurrent`  
*The current available receive buffer descriptor pointer.*
- volatile `enet_tx_bd_struct_t * txBdBase`  
*Transmit buffer descriptor base address pointer.*
- volatile `enet_tx_bd_struct_t * txBdCurrent`  
*The current available transmit buffer descriptor pointer.*
- `uint32_t rxBuffSizeAlign`  
*Receive buffer size alignment.*
- `uint32_t txBuffSizeAlign`  
*Transmit buffer size alignment.*

- `enet_callback_t callback`  
*Callback function.*
- `void *userData`  
*Callback function parameter.*

#### 11.3.7.0.0.13 Field Documentation

11.3.7.0.0.13.1 `volatile enet_rx_bd_struct_t* enet_handle_t::rxBdBase`

11.3.7.0.0.13.2 `volatile enet_rx_bd_struct_t* enet_handle_t::rxBdCurrent`

11.3.7.0.0.13.3 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdBase`

11.3.7.0.0.13.4 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdCurrent`

11.3.7.0.0.13.5 `uint32_t enet_handle_t::rxBuffSizeAlign`

11.3.7.0.0.13.6 `uint32_t enet_handle_t::txBuffSizeAlign`

11.3.7.0.0.13.7 `enet_callback_t enet_handle_t::callback`

11.3.7.0.0.13.8 `void* enet_handle_t::userData`

### 11.4 Macro Definition Documentation

11.4.1 `#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

Version 2.1.1.

## Macro Definition Documentation

- 11.4.2 `#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U`
- 11.4.3 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U`
- 11.4.4 `#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U`
- 11.4.5 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U`
- 11.4.6 `#define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U`
- 11.4.7 `#define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U`
- 11.4.8 `#define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U`
- 11.4.9 `#define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U`
- 11.4.10 `#define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK 0x0020U`
- 11.4.11 `#define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U`
- 11.4.12 `#define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U`
- 11.4.13 `#define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U`
- 11.4.14 `#define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U`
- 11.4.15 `#define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U`
- 11.4.16 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK 0x4000U`
- 11.4.17 `#define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U`
- 11.4.18 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK 0x1000U`
- 11.4.19 `#define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U`
- 11.4.20 `#define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U`
- 11.4.21 `#define ENET_BUFFDESCRIPTOR_RX_ERR_MASK`

## Enumeration Type Documentation

```
(ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK |  
ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK | \  
ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK |  
ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK |  
ENET_BUFFDESCRIPTOR_RX_CRC_MASK)
```

**11.4.22 #define ENET\_FRAME\_MAX\_FRAMELEN 1518U**

**11.4.23 #define ENET\_FIFO\_MIN\_RX\_FULL 5U**

**11.4.24 #define ENET\_RX\_MIN\_BUFFERSIZE 256U**

**11.4.25 #define ENET\_PHY\_MAXADDRESS (ENET\_MMFR\_PA\_MASK >>  
ENET\_MMFR\_PA\_SHIFT)**

## 11.5 Typedef Documentation

**11.5.1 typedef void(\* enet\_callback\_t)(ENET\_Type \*base, enet\_handle\_t \*handle,  
enet\_event\_t event, void \*userData)**

## 11.6 Enumeration Type Documentation

### 11.6.1 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\_TxFrameBusy* Transmit buffer descriptors are under process.

*kStatus\_ENET\_TxFrameFail* Transmit frame fail.

### 11.6.2 enum enet\_mii\_mode\_t

Enumerator

*kENET\_MiiMode* MII mode for data interface.

*kENET\_RmiiMode* RMII mode for data interface.

### 11.6.3 enum enet\_mii\_speed\_t

Enumerator

*kENET\_MiiSpeed10M* Speed 10 Mbps.

***kENET\_MiiSpeed100M*** Speed 100 Mbps.

#### 11.6.4 enum enet\_mii\_duplex\_t

Enumerator

***kENET\_MiiHalfDuplex*** Half duplex mode.

***kENET\_MiiFullDuplex*** Full duplex mode.

#### 11.6.5 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.

#### 11.6.6 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.

#### 11.6.7 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.

#### 11.6.8 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](#).

## Enumeration Type Documentation

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 VLAN tag frame.

### 11.6.9 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 interrupt source.
- kENET\_WakeupInterrupt* WAKEUP interrupt source.
- kENET\_TsAvailInterrupt* TS AVAIL interrupt source for PTP.
- kENET\_TsTimerInterrupt* TS WRAP interrupt source for PTP.

### 11.6.10 enum enet\_event\_t

Enumerator

- kENET\_RxEvent* Receive event.
- kENET\_TxEvent* Transmit event.
- kENET\_ErrEvent* Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

*kENET\_WakeUpEvent* Wake up from sleep mode event.

*kENET\_TimeStampEvent* Time stamp event.

*kENET\_TimeStampAvailEvent* Time stamp available event.

### 11.6.11 enum enet\_tx\_accelerator\_t

Enumerator

*kENET\_TxAccelIsShift16Enabled* Transmit FIFO shift-16.

*kENET\_TxAccelIpCheckEnabled* Insert IP header checksum.

*kENET\_TxAccelProtoCheckEnabled* Insert protocol checksum.

### 11.6.12 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.

## 11.7 Function Documentation

### 11.7.1 void ENET\_GetDefaultConfig ( enet\_config\_t \* config )

The purpose of this API is to get the default ENET MAC controller configuration structure for [ENET\\_Init\(\)](#). Users may use the initialized structure unchanged in [ENET\\_Init\(\)](#) or modify fields of the structure before calling [ENET\\_Init\(\)](#). This is an example.

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Parameters

<i>config</i>	The ENET mac controller configuration structure pointer.
---------------	--

### 11.7.2 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.
<i>macAddr</i>	ENET mac address of the 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\(\)](#).

### 11.7.3 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.
-------------	-------------------------------

### 11.7.4 static void ENET\_Reset( ENET\_Type \* *base* ) [inline], [static]

This function restores the ENET module to the reset state. Note that this function sets all registers to the reset state. As a result, the ENET module can't work after calling this function.

Parameters

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

### 11.7.5 void ENET\_SetMII ( ENET\_Type \* *base*, enet\_mii\_speed\_t *speed*, enet\_mii\_duplex\_t *duplex* )

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.

### 11.7.6 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>

### 11.7.7 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

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

Returns

The SMI setup status true or false.

## Function Documentation

11.7.8 **static uint32\_t ENET\_ReadSMIData ( ENET\_Type \* *base* ) [inline],  
[static]**

Parameters

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

Returns

The data read from PHY

### 11.7.9 void ENET\_StartSMIRead ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_read\_t *operation* )

Parameters

<i>base</i>	ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register.
<i>operation</i>	The read operation.

### 11.7.10 void ENET\_StartSMIWrite ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_write\_t *operation*, uint32\_t *data* )

Parameters

<i>base</i>	ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register.
<i>operation</i>	The write operation.
<i>data</i>	The data written to PHY.

### 11.7.11 void ENET\_StartExtC45SMIRead ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg* )

## Function Documentation

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): phyReg = (devaddr << 16)   regAddr.

### 11.7.12 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): phyReg = (devaddr << 16)   regAddr.
<i>data</i>	The data written to PHY.

### 11.7.13 void ENET\_SetMacAddr ( ENET\_Type \* *base*, uint8\_t \* *macAddr* )

Parameters

<i>base</i>	ENET peripheral base address.
<i>macAddr</i>	The six-byte Mac address pointer. The pointer is allocated by application and input into the API.

### 11.7.14 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.

**11.7.15 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.

**11.7.16 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.

**11.7.17 static void ENET\_ActiveRead ( ENET\_Type \* *base* ) [inline], [static]**

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.

**11.7.18 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.

## Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>enable</i>	True enable sleep mode, false disable sleep mode.

### 11.7.19 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 as the mask to get the exact the accelerator option.
<i>rxAccelOption</i>	The receive accelerator option. The "enet_rx_accelerator_t" is recommended as the mask to get the exact the accelerator option.

### 11.7.20 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.

```
*     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 :: enet_interrupt_enable_t.

### 11.7.21 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 :: enet_interrupt_enable_t.

### 11.7.22 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.

### 11.7.23 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.

```
*     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 :: enet_interrupt_enable_t.

## Function Documentation

**11.7.24 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.

**11.7.25 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.

**11.7.26 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".

## Function Documentation

### Parameters

<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>length</i>	The length of the valid frame received.

### Return values

<i>kStatus_ENET_RxFrameEmpty</i>	No frame received. Should not call ENET_ReadFrame to read frame.
<i>kStatus_ENET_RxFrameError</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.

### 11.7.27 **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.

**11.7.28 `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_TxFrameBusy</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> .

**11.7.29 `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.

**11.7.30 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.

**11.7.31 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.

**11.7.32 void ENET\_CommonFrame0IRQHandler ( ENET\_Type \* *base* )**

This is used for the combined tx/rx/error interrupt for single ring (ring 0).

Parameters

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

# Chapter 12

## EPIT: Enhanced Periodic Interrupt Timer

### 12.1 Overview

The MCUXpresso SDK provides a driver for the Enhanced Periodic Interrupt Timer (EPIT) of MCUXpresso SDK devices.

### 12.2 Function groups

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

#### 12.2.1 Initialization and deinitialization

The function `EPIT_Init()` initializes the epit with specified configurations. The function `EPIT_GetDefaultConfig()` gets the default configurations. The initialization function configures the restart/free-run mode and input selection when running.

The function `EPIT_Deinit()` stops the timer and turns off the module clock.

### 12.3 Typical use case

#### 12.3.1 EPIT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second.

```
int main(void)
{
    /* Structure of initialize EPIT */
    epit_config_t epitConfig;

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    EPIT_GetDefaultConfig(&epitConfig);

    /* Init EPIT module */
    EPIT_Init(EXAMPLE_EPIT, &epitConfig);

    /* Set timer period */
    EPIT_SetTimerPeriod(EXAMPLE_EPIT, USEC_TO_COUNT(1000000U, EXAMPLE_EPIT_CLK_FREQ) - 1);
    EPIT_SetOutputCompareValue(EXAMPLE_EPIT, 0);

    /* Enable output compare interrupts */
    EPIT_EnableInterrupts(EXAMPLE_EPIT,
        kEPIT_OutputCompareInterruptEnable);

    /* Enable at the Interrupt */
}
```

## Typical use case

```
EnableIRQ(EPIT IRQ_ID);

/* Start Timer */
PRINTF("\r\nStarting EPIT timer ...");
EPIT_StartTimer(EXAMPLE_EPIT);

while (true)
{
    /* Check whether occur interrupt and toggle LED */
    if (true == epitIsrFlag)
    {
        PRINTF("\r\n EPIT interrupt is occurred !");
        epitIsrFlag = false;
    }
    else
    {
        __WFI();
    }
}
```

## Data Structures

- struct `epit_config_t`  
*Structure to configure the running mode.* [More...](#)

## Enumerations

- enum `epit_clock_source_t` {  
    kEPIT\_ClockSource\_Off = 0U,  
    kEPIT\_ClockSource\_Periph = 1U,  
    kEPIT\_ClockSource\_HighFreq = 2U,  
    kEPIT\_ClockSource\_LowFreq = 3U }  
*List of clock sources.*
- enum `epit_output_operation_mode_t` {  
    kEPIT\_OutputOperation\_Disconnected = 0U,  
    kEPIT\_OutputOperation\_Toggle = 1U,  
    kEPIT\_OutputOperation\_Clear = 2U,  
    kEPIT\_OutputOperation\_Set = 3U }  
*List of output compare operation mode.*
- enum `epit_interrupt_enable_t` { kEPIT\_OutputCompareInterruptEnable = EPIT\_CR\_OCIEN\_MSK }
- enum `epit_status_flags_t` { kEPIT\_OutputCompareFlag = EPIT\_SR\_OCIF\_MASK }  
*List of EPIT status flags.*

## Driver version

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

## Software Reset

- static void `EPIT_SoftwareReset` (EPIT\_Type \*base)  
*Software reset of EPIT module.*

## Initialization and deinitialization

- void **EPIT\_Init** (EPIT\_Type \*base, const **epit\_config\_t** \*config)  
*Ungates the EPIT clock and configures the peripheral for a basic operation.*
- void **EPIT\_Deinit** (EPIT\_Type \*base)  
*Disables the module and gates the EPIT clock.*
- void **EPIT\_GetDefaultConfig** (**epit\_config\_t** \*config)  
*Fills in the EPIT configuration structure with default settings.*

## Clock source and frequency control

- static void **EPIT\_SetClockSource** (EPIT\_Type \*base, **epit\_clock\_source\_t** source)  
*Set clock source of EPIT.*
- static void **EPIT\_SetClockDivider** (EPIT\_Type \*base, uint32\_t divider)  
*Set clock divider inside EPIT module.*
- static uint32\_t **EPIT\_GetClockDivider** (EPIT\_Type \*base)  
*Get clock divider inside EPIT module.*

## Timer Start and Stop

- static void **EPIT\_StartTimer** (EPIT\_Type \*base)  
*Start EPIT timer.*
- static void **EPIT\_StopTimer** (EPIT\_Type \*base)  
*Stop EPIT timer.*

## Read and Write the timer period

- static void **EPIT\_SetTimerPeriod** (EPIT\_Type \*base, uint32\_t ticks)  
*Sets the timer period in units of count.*
- static uint32\_t **EPIT\_GetCurrentTimerCount** (EPIT\_Type \*base)  
*Reads the current timer counting value.*

## Output Signal Control

- static void **EPIT\_SetOutputOperationMode** (EPIT\_Type \*base, **epit\_output\_operation\_mode\_t** mode)  
*Set EPIT output compare operation mode.*
- static void **EPIT\_SetOutputCompareValue** (EPIT\_Type \*base, uint32\_t value)  
*Set EPIT output compare value.*

## Interrupt Interface

- static void **EPIT\_EnableInterrupts** (EPIT\_Type \*base, uint32\_t mask)  
*Enables the selected EPIT interrupts.*
- static void **EPIT\_DisableInterrupts** (EPIT\_Type \*base, uint32\_t mask)  
*Disables the selected EPIT interrupts.*
- static uint32\_t **EPIT\_GetEnabledInterrupts** (EPIT\_Type \*base)  
*Gets the enabled EPIT interrupts.*

### Status Interface

- static uint32\_t [EPIT\\_GetStatusFlags](#) (EPIT\_Type \*base)  
*Gets the EPIT status flags.*
- static void [EPIT\\_ClearStatusFlags](#) (EPIT\_Type \*base, uint32\_t mask)  
*Clears the EPIT status flags.*

## 12.4 Data Structure Documentation

### 12.4.1 struct epit\_config\_t

#### Data Fields

- [epit\\_clock\\_source\\_t clockSource](#)  
*clock source for EPIT module.*
- uint32\_t [divider](#)  
*clock divider (prescaler+1) from clock source to counter.*
- bool [enableRunInStop](#)  
*EPIT enabled in stop mode.*
- bool [enableRunInWait](#)  
*EPIT enabled in wait mode.*
- bool [enableRunInDbg](#)  
*EPIT enabled in debug mode.*
- bool [enableCounterOverwrite](#)  
*set timer period results in counter value being overwritten.*
- bool [enableFreeRun](#)  
*true: free-running mode, counter will be reset to 0xFFFFFFFF when timer expires; false: set-and-forget mode, counter will be reloaded from set timer periods.*
- bool [enableResetMode](#)  
*true: counter is reset to timer periods in set-and-forget mode or 0xFFFFFFFF in free-running mode when enabled; false: counter restores the value that it was disabled when enabled.*

#### 12.4.1.0.0.14 Field Documentation

##### 12.4.1.0.0.14.1 [epit\\_clock\\_source\\_t epit\\_config\\_t::clockSource](#)

##### 12.4.1.0.0.14.2 [uint32\\_t epit\\_config\\_t::divider](#)

##### 12.4.1.0.0.14.3 [bool epit\\_config\\_t::enableRunInStop](#)

##### 12.4.1.0.0.14.4 [bool epit\\_config\\_t::enableRunInWait](#)

##### 12.4.1.0.0.14.5 [bool epit\\_config\\_t::enableRunInDbg](#)

##### 12.4.1.0.0.14.6 [bool epit\\_config\\_t::enableCounterOverwrite](#)

##### 12.4.1.0.0.14.7 [bool epit\\_config\\_t::enableFreeRun](#)

##### 12.4.1.0.0.14.8 [bool epit\\_config\\_t::enableResetMode](#)

## 12.5 Enumeration Type Documentation

### 12.5.1 enum epit\_clock\_source\_t

Note

Actual number of clock sources is SoC dependent

Enumerator

*kEPIT\_ClockSource\_Off* EPIT Clock Source Off.

*kEPIT\_ClockSource\_Pерiph* EPIT Clock Source from Peripheral Clock.

*kEPIT\_ClockSource\_HighFreq* EPIT Clock Source from High Frequency Reference Clock.

*kEPIT\_ClockSource\_LowFreq* EPIT Clock Source from Low Frequency Reference Clock.

### 12.5.2 enum epit\_output\_operation\_mode\_t

Enumerator

*kEPIT\_OutputOperation\_Disconnected* EPIT Output Operation: Disconnected from pad.

*kEPIT\_OutputOperation\_Toggle* EPIT Output Operation: Toggle output pin.

*kEPIT\_OutputOperation\_Clear* EPIT Output Operation: Clear output pin.

*kEPIT\_OutputOperation\_Set* EPIT Output Operation: Set output pin.

### 12.5.3 enum epit\_interrupt\_enable\_t

Enumerator

*kEPIT\_OutputCompareInterruptEnable* Output Compare interrupt enable.

### 12.5.4 enum epit\_status\_flags\_t

Enumerator

*kEPIT\_OutputCompareFlag* Output Compare flag.

## 12.6 Function Documentation

### 12.6.1 static void EPIT\_SoftwareReset( EPIT\_Type \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

### 12.6.2 void EPIT\_Init ( EPIT\_Type \* *base*, const epit\_config\_t \* *config* )

This function issues a software reset to reset all the registers to their reset values, except for the EN, ENMOD, STOPEN, WAITEN and DBGEN bits in Control register.

Note

This API should be called at the beginning of the application using the EPIT driver.

Parameters

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

### 12.6.3 void EPIT\_Deinit ( EPIT\_Type \* *base* )

Parameters

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

### 12.6.4 void EPIT\_GetDefaultConfig ( epit\_config\_t \* *config* )

The default values are:

```
* config->clockSource = kEPIT_ClockSource_Periph;
* config->divider = 1U;
* config->enableRunInStop = true;
* config->enableRunInWait = true;
* config->enableRunInDbg = false;
* config->enableCounterOverwrite = false;
* config->enableFreeRun = false;
* config->enableResetMode = true;
*
```

Parameters

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

### 12.6.5 static void EPIT\_SetClockSource ( EPIT\_Type \* *base*, epit\_clock\_source\_t *source* ) [inline], [static]

Parameters

<i>base</i>	EPIT peripheral base address.
<i>source</i>	clock source to switch to.

### 12.6.6 static void EPIT\_SetClockDivider ( EPIT\_Type \* *base*, uint32\_t *divider* ) [inline], [static]

Parameters

<i>base</i>	EPIT peripheral base address.
<i>divider</i>	Clock divider in EPIT module (1-4096, divider = prescaler + 1).

### 12.6.7 static uint32\_t EPIT\_GetClockDivider ( EPIT\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

clock divider in EPIT module (1-4096).

### 12.6.8 static void EPIT\_StartTimer ( EPIT\_Type \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

### 12.6.9 static void EPIT\_StopTimer( EPIT\_Type \* *base* ) [inline], [static]

Parameters

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

### 12.6.10 static void EPIT\_SetTimerPeriod( EPIT\_Type \* *base*, uint32\_t *ticks* ) [inline], [static]

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. When enableCounterOverwrite is false, writing a new value to this register does not restart the timer, and the value is loaded after the timer expires. When enableCounterOverwrite is true, the counter will be set immediately and starting counting down from that value.

Note

User can call the utility macros provided in `fsl_common.h` to convert to ticks.

Parameters

<i>base</i>	EPIT peripheral base address.
<i>ticks</i>	Timer period in units of ticks.

### 12.6.11 static uint32\_t EPIT\_GetCurrentTimerCount( EPIT\_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 microseconds or milliseconds.

Parameters

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

Returns

Current timer counting value in ticks.

#### 12.6.12 static void EPIT\_SetOutputOperationMode ( EPIT\_Type \* *base*, epit\_output\_operation\_mode\_t *mode* ) [inline], [static]

Parameters

<i>base</i>	EPIT peripheral base address.
<i>mode</i>	EPIT output compare operation mode.

#### 12.6.13 static void EPIT\_SetOutputCompareValue ( EPIT\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

<i>base</i>	EPIT peripheral base address.
<i>value</i>	EPIT output compare value.

#### 12.6.14 static void EPIT\_EnableInterrupts ( EPIT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	EPIT peripheral base address.
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">epit_interrupt_enable_t</a>

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

## Function Documentation

Parameters

<i>base</i>	EPIT peripheral base address
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">epit_interrupt_enable_t</a>

**12.6.16 static uint32\_t EPIT\_GetEnabledInterrupts ( EPIT\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [epit\\_interrupt\\_enable\\_t](#)

**12.6.17 static uint32\_t EPIT\_GetStatusFlags ( EPIT\_Type \* *base* ) [inline],  
[static]**

Parameters

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

Returns

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

**12.6.18 static void EPIT\_ClearStatusFlags ( EPIT\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

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

## Function Documentation

# Chapter 13

## FlexCAN: Flex Controller Area Network Driver

### 13.1 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](#)

## FlexCAN Driver

### 13.2 FlexCAN Driver

#### 13.2.1 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.

#### 13.2.2 Typical use case

##### 13.2.2.1 Message Buffer Send Operation

```
flexcan_config_t flexcanConfig;
flexcan_frame_t txFrame;

/* Init FlexCAN module. */
FLEXCAN_SetDefaultConfig(&flexcanConfig);
FLEXCAN_Init(EXAMPLE_CAN, &flexcanConfig);

/* Enable FlexCAN module. */
FLEXCAN_Enable(EXAMPLE_CAN, true);

/* Sets up the transmit message buffer. */
FLEXCAN_SetTxMbConfig(EXAMPLE_CAN, TX_MESSAGE_BUFFER_INDEX, true);

/* Prepares the transmit frame for sending. */
txFrame.format = KFLEXCAN_FrameFormatStandard;
txFrame.type   = KFLEXCAN_FrameTypeData;
txFrame.id     = FLEXCAN_ID_STD(0x123);
txFrame.length = 8;
txFrame.dataWord0 = CAN_WORD0_DATA_BYTE_0(0x11) |
                   CAN_WORD0_DATA_BYTE_1(0x22) |
                   CAN_WORD0_DATA_BYTE_2(0x33) |
                   CAN_WORD0_DATA_BYTE_3(0x44);
txFrame.dataWord1 = CAN_WORD1_DATA_BYTE_4(0x55) |
                   CAN_WORD1_DATA_BYTE_5(0x66) |
                   CAN_WORD1_DATA_BYTE_6(0x77) |
                   CAN_WORD1_DATA_BYTE_7(0x88);
/* Writes a transmit message buffer to send a CAN Message. */
FLEXCAN_WriteTxMb(EXAMPLE_CAN, TX_MESSAGE_BUFFER_INDEX, &txFrame);

/* Waits until the transmit message buffer is empty. */
while (!FLEXCAN_GetMbStatusFlags(EXAMPLE_CAN, 1 << TX_MESSAGE_BUFFER_INDEX));

/* Cleans the transmit message buffer empty status. */
FLEXCAN_ClearMbStatusFlags(EXAMPLE_CAN, 1 << TX_MESSAGE_BUFFER_INDEX);
```

##### 13.2.2.2 Message Buffer Receive Operation

```
flexcan_config_t flexcanConfig;
flexcan_frame_t rxFrame;

/* Initializes the FlexCAN module. */
FLEXCAN_SetDefaultConfig(&flexcanConfig);
FLEXCAN_Init(EXAMPLE_CAN, &flexcanConfig);

/* Enables the FlexCAN module. */
FLEXCAN_Enable(EXAMPLE_CAN, true);

/* Sets up the receive message buffer. */
```

```

mbConfig.format      = KFLEXCAN_FrameFormatStandard;
mbConfig.type        = KFLEXCAN_FrameTypeData;
mbConfig.id          = FLEXCAN_ID_STD(0x123);
FLEXCAN_SetRxMbConfig(EXAMPLE_CAN, RX_MESSAGE_BUFFER_INDEX, &mbConfig, true);

/* Waits until the receive message buffer is full. */
while (!FLEXCAN_GetMbStatusFlags(EXAMPLE_CAN, 1 << RX_MESSAGE_BUFFER_INDEX));

/* Reads the received message from the receive message buffer. */
FLEXCAN_ReadRxMb(EXAMPLE_CAN, RX_MESSAGE_BUFFER_INDEX, &rxFrame);

/* Cleans the receive message buffer full status. */
FLEXCAN_ClearMbStatusFlags(EXAMPLE_CAN, 1 << RX_MESSAGE_BUFFER_INDEX);

```

### 13.2.2.3 Receive FIFO Operation

```

uint32_t rxFifoFilter[] = {FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x321, 0, 0),
                           FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x321, 1, 0),
                           FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x123, 0, 0),
                           FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x123, 1, 0)};
;

flexcan_config_t flexcanConfig;
flexcan_frame_t rxFrame;

/* Initializes the FlexCAN module. */
FLEXCAN_GetDefaultConfig(&flexcanConfig);
FLEXCAN_Init(EXAMPLE_CAN, &flexcanConfig);

/* Enables the FlexCAN module. */
FLEXCAN_Enable(EXAMPLE_CAN, true);

/* Sets up the receive FIFO. */
rxFifoConfig.idFilterTable = rxFifoFilter;
rxFifoConfig.idFilterType  = KFLEXCAN_RxFifoFilterTypeA;
rxFifoConfig.idFilterNum   = sizeof(rxFifoFilter) / sizeof(rxFifoFilter[0]);
rxFifoConfig.priority     = KFLEXCAN_RxFifoPrioHigh;
FLEXCAN_SetRxFifoConfig(EXAMPLE_CAN, &rxFifoConfig, true);

/* Waits until the receive FIFO becomes available. */
while (!FLEXCAN_GetMbStatusFlags(EXAMPLE_CAN, KFLEXCAN_RxFifoFrameAvlFlag));

/* Reads the message from the receive FIFO. */
FLEXCAN_ReadRxFifo(EXAMPLE_CAN, &rxFrame);

/* Cleans the receive FIFO available status. */
FLEXCAN_ClearMbStatusFlags(EXAMPLE_CAN, KFLEXCAN_RxFifoFrameAvlFlag);

```

## Data Structures

- struct [flexcan\\_frame\\_t](#)  
*FlexCAN message frame structure.* [More...](#)
- struct [flexcan\\_config\\_t](#)  
*FlexCAN module configuration structure.* [More...](#)
- struct [flexcan\\_timing\\_config\\_t](#)  
*FlexCAN protocol timing characteristic configuration structure.* [More...](#)
- struct [flexcan\\_rx\\_mb\\_config\\_t](#)  
*FlexCAN Receive Message Buffer configuration structure.* [More...](#)
- struct [flexcan\\_rx\\_fifo\\_config\\_t](#)

## FlexCAN Driver

*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)
 

*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) **FLEXCAN\_RX\_FIFO\_STD\_MASK\_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) **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_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` {

```

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 `FLEXCAN_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 0))  
*FlexCAN driver version 2.2.0.*

## Initialization and deinitialization

- 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.*

## FlexCAN Driver

### 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\_ReadRxFifo** (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)  
*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.*

### 13.2.3 Data Structure Documentation

#### 13.2.3.1 struct flexcan\_frame\_t

##### 13.2.3.1.0.15 Field Documentation

13.2.3.1.0.15.1 uint32\_t flexcan\_frame\_t::timestamp

13.2.3.1.0.15.2 uint32\_t flexcan\_frame\_t::length

13.2.3.1.0.15.3 uint32\_t flexcan\_frame\_t::type

13.2.3.1.0.15.4 uint32\_t flexcan\_frame\_t::format

13.2.3.1.0.15.5 uint32\_t flexcan\_frame\_t::\_\_pad0\_\_

13.2.3.1.0.15.6 uint32\_t flexcan\_frame\_t::idhit

13.2.3.1.0.15.7 uint32\_t flexcan\_frame\_t::id

13.2.3.1.0.15.8 uint32\_t flexcan\_frame\_t::dataWord0

13.2.3.1.0.15.9 uint32\_t flexcan\_frame\_t::dataWord1

13.2.3.1.0.15.10 uint8\_t flexcan\_frame\_t::dataByte3

13.2.3.1.0.15.11 uint8\_t flexcan\_frame\_t::dataByte2

13.2.3.1.0.15.12 uint8\_t flexcan\_frame\_t::dataByte1

13.2.3.1.0.15.13 uint8\_t flexcan\_frame\_t::dataByte0

13.2.3.1.0.15.14 uint8\_t flexcan\_frame\_t::dataByte7

13.2.3.1.0.15.15 uint8\_t flexcan\_frame\_t::dataByte6

13.2.3.1.0.15.16 uint8\_t flexcan\_frame\_t::dataByte5

13.2.3.1.0.15.17 uint8\_t flexcan\_frame\_t::dataByte4

#### 13.2.3.2 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.*

### 13.2.3.2.0.16 Field Documentation

13.2.3.2.0.16.1 `uint32_t flexcan_config_t::baudRate`

13.2.3.2.0.16.2 `uint8_t flexcan_config_t::maxMbNum`

13.2.3.2.0.16.3 `bool flexcan_config_t::enableLoopBack`

13.2.3.2.0.16.4 `bool flexcan_config_t::enableSelfWakeup`

13.2.3.2.0.16.5 `bool flexcan_config_t::enableIndividMask`

### 13.2.3.3 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.*
- `uint8_t phaseSeg2`  
*Phase Segment 2.*
- `uint8_t propSeg`  
*Propagation Segment.*

### 13.2.3.3.0.17 Field Documentation

13.2.3.3.0.17.1 `uint16_t flexcan_timing_config_t::preDivider`

13.2.3.3.0.17.2 `uint8_t flexcan_timing_config_t::rJumpwidth`

13.2.3.3.0.17.3 `uint8_t flexcan_timing_config_t::phaseSeg1`

13.2.3.3.0.17.4 `uint8_t flexcan_timing_config_t::phaseSeg2`

13.2.3.3.0.17.5 `uint8_t flexcan_timing_config_t::propSeg`

### 13.2.3.4 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.

## FlexCAN Driver

### 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 or Extend).*
- `flexcan_frame_type_t type`  
*CAN Frame Type(Data or Remote).*

### 13.2.3.4.0.18 Field Documentation

#### 13.2.3.4.0.18.1 `uint32_t flexcan_rx_mb_config_t::id`

#### 13.2.3.4.0.18.2 `flexcan_frame_format_t flexcan_rx_mb_config_t::format`

#### 13.2.3.4.0.18.3 `flexcan_frame_type_t flexcan_rx_mb_config_t::type`

### 13.2.3.5 `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.*

### 13.2.3.5.0.19 Field Documentation

#### 13.2.3.5.0.19.1 `uint32_t* flexcan_rx_fifo_config_t::idFilterTable`

#### 13.2.3.5.0.19.2 `uint8_t flexcan_rx_fifo_config_t::idFilterNum`

#### 13.2.3.5.0.19.3 `flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType`

#### 13.2.3.5.0.19.4 `flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority`

### 13.2.3.6 `struct flexcan_mb_transfer_t`

### Data Fields

- `flexcan_frame_t * frame`  
*The buffer of CAN Message to be transfer.*
- `uint8_t mbIdx`  
*The index of Message buffer used to transfer Message.*

### 13.2.3.6.0.20 Field Documentation

#### 13.2.3.6.0.20.1 `flexcan_frame_t* flexcan_mb_transfer_t::frame`

#### 13.2.3.6.0.20.2 `uint8_t flexcan_mb_transfer_t::mbIdx`

### 13.2.3.7 `struct flexcan_fifo_transfer_t`

#### Data Fields

- `flexcan_frame_t * frame`  
*The buffer of CAN Message to be received from Rx FIFO.*

### 13.2.3.7.0.21 Field Documentation

#### 13.2.3.7.0.21.1 `flexcan_frame_t* flexcan_fifo_transfer_t::frame`

### 13.2.3.8 `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.*

## FlexCAN Driver

### 13.2.3.8.0.22 Field Documentation

13.2.3.8.0.22.1 `flexcan_transfer_callback_t flexcan_handle_t::callback`

13.2.3.8.0.22.2 `void* flexcan_handle_t::userData`

13.2.3.8.0.22.3 `flexcan_frame_t* volatile flexcan_handle_t::mbFrameBuf[CAN_WORD1_COUNT]`

13.2.3.8.0.22.4 `flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf`

13.2.3.8.0.22.5 `volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]`

13.2.3.8.0.22.6 `volatile uint8_t flexcan_handle_t::rxFifoState`

### 13.2.4 Macro Definition Documentation

13.2.4.1 `#define FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

13.2.4.2 `#define FLEXCAN_ID_STD( id ) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT) & CAN_ID_STD_MASK)`

Standard Frame ID helper macro.

13.2.4.3 `#define FLEXCAN_ID_EXT( id )`

**Value:**

```
((uint32_t)((uint32_t)(id)) << CAN_ID_EXT_SHIFT) & \  
 (CAN_ID_EXT_MASK | CAN_ID_STD_MASK)
```

13.2.4.4 `#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.

13.2.4.5 `#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))
```

**13.2.4.6 #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.

**13.2.4.7 #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))
```

**13.2.4.8 #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))
```

**13.2.4.9 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH( id ) (((uint32\_t)(id) & 0x7F8) << 21)****13.2.4.10 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH( id ) (((uint32\_t)(id) & 0x7F8) << 13)****13.2.4.11 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW( id ) (((uint32\_t)(id) & 0x7F8) << 5)****13.2.4.12 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW( id ) (((uint32\_t)(id) & 0x7F8) >> 3)****13.2.4.13 #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))
```

## FlexCAN Driver

**13.2.4.14 #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))
```

**13.2.4.15 #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))
```

**13.2.4.16 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH( id )**  
**((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) << 3)**

**13.2.4.17 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH( id )**

**Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 5) \
```

**13.2.4.18 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW( id )**

**Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 13) \
```

**13.2.4.19 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW( id )**  
**((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) >> 21)**

**13.2.4.20 #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.

**13.2.4.21 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(           \
    id, rtr, ide)
```

**13.2.4.22 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(           \
    id, rtr, ide)
```

**13.2.4.23 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH( *id* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(           \
    id)
```

**13.2.4.24 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(           \
    id)
```

**13.2.4.25 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW( *id* )****Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(           \
    id)
```

**13.2.4.26 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW( *id* )  
 ) FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW(id)**

\

## FlexCAN Driver

- 13.2.4.27 `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A( id, rtr, ide ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)`
- 13.2.4.28 `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH( id, rtr, ide )`

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(           \
    id, rtr, ide)
```

- 13.2.4.29 `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW( id, rtr, ide )`

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(           \
    id, rtr, ide)
```

- 13.2.4.30 `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH( id ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)`

\

- 13.2.4.31 `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH( id )`

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(       \
    id)
```

- 13.2.4.32 `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW( id )`

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(        \
    id)
```

```
13.2.4.33 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW( id
    ) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)
```

## 13.2.5 Typedef Documentation

**13.2.5.1 `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.

## 13.2.6 Enumeration Type Documentation

### 13.2.6.1 `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.

### 13.2.6.2 `enum flexcan_frame_format_t`

Enumerator

- kFLEXCAN\_FrameFormatStandard*** Standard frame format attribute.
- kFLEXCAN\_FrameFormatExtend*** Extend frame format attribute.

### 13.2.6.3 enum flexcan\_frame\_type\_t

Enumerator

***kFLEXCAN\_FrameTypeData*** Data frame type attribute.

***kFLEXCAN\_FrameTypeRemote*** Remote frame type attribute.

### 13.2.6.4 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.

### 13.2.6.5 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.

### 13.2.6.6 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.

### 13.2.6.7 enum \_flexcan\_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions. Note: The CPU read action clears FLEXCAN\_ErrorFlag, therefore user need to read FLEXCAN\_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.

### 13.2.6.8 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.

### 13.2.6.9 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.

## FlexCAN Driver

*kFLEXCAN\_RxFifoWarningFlag* Rx FIFO almost full flag.  
*kFLEXCAN\_RxFifoFrameAvlFlag* Frames available in Rx FIFO flag.

### 13.2.7 Function Documentation

#### 13.2.7.1 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        = 125000U;
*     flexcanConfig.maxMbNum        = 16;
*     flexcanConfig.enableLoopBack  = false;
*     flexcanConfig.enableSelfWakeup = false;
*     flexcanConfig.enableIndividMask = false;
*     flexcanConfig.enableDoze      = false;
*     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.

#### 13.2.7.2 void FLEXCAN\_Deinit ( CAN\_Type \* *base* )

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

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

#### 13.2.7.3 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* = 125000U; *flexcanConfig->maxMbNum* = 16; *flexcanConfig->enableLoopBack* = false; *flexcanConfig->enableSelfWakeup* = false; *flexcanConfig->enableIndividMask* = false; *flexcanConfig->enableDoze* = false;

Parameters

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

#### 13.2.7.4 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.

#### 13.2.7.5 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.

#### 13.2.7.6 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.

## FlexCAN Driver

### 13.2.7.7 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.

### 13.2.7.8 void FLEXCAN\_SetTxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mbIdx</i>	The Message Buffer index.
<i>enable</i>	Enable/disable Tx Message Buffer. <ul style="list-style-type: none"><li>• true: Enable Tx Message Buffer.</li><li>• false: Disable Tx Message Buffer.</li></ul>

### 13.2.7.9 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>

### 13.2.7.10 void FLEXCAN\_SetRx\_fifoConfig ( 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>

### 13.2.7.11 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.
-------------	----------------------------------

## FlexCAN Driver

Returns

FlexCAN status flags which are ORed by the enumerators in the \_flexcan\_flags.

**13.2.7.12 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 <a href="#">_flexcan_flags</a> .

### 13.2.7.13 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.

### 13.2.7.14 static uint64\_t FLEXCAN\_GetMbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function gets the interrupt flags of a given Message Buffers.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mask</i>	The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

### 13.2.7.15 static void FLEXCAN\_ClearMbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function clears 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.

### 13.2.7.16 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> .

### 13.2.7.17 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 <a href="#">_flexcan_interrupt_enable</a> .

### 13.2.7.18 static void FLEXCAN\_EnableMbInterrupts ( 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.

**13.2.7.19 static void FLEXCAN\_DisableMbInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* )  
[inline], [static]**

This function disables the interrupts of given Message Buffers.

## FlexCAN Driver

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>mask</i>	The ORed FlexCAN Message Buffer mask.

**13.2.7.20 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.

**13.2.7.21 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.

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.

**13.2.7.22 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.

### 13.2.7.23 **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.

### 13.2.7.24 **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.
-------------	----------------------------------

## FlexCAN Driver

<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.

### 13.2.7.25 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.
<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.

### 13.2.7.26 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.

### 13.2.7.27 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.

### 13.2.7.28 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.

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.

## FlexCAN Driver

**13.2.7.29 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.

### 13.2.7.30 **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.

### 13.2.7.31 **void FLEXCAN\_TransferAbortSend ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )**

This function aborts the interrupt driven message send process.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>mbIdx</i>	The FlexCAN Message Buffer index.

## FlexCAN Driver

**13.2.7.32 void FLEXCAN\_TransferAbortReceive ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )**

This function aborts the interrupt driven message receive process.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	FlexCAN handle pointer.
<i>mbIdx</i>	The FlexCAN Message Buffer index.

### 13.2.7.33 void FLEXCAN\_TransferAbortReceiveFifo ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

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

### 13.2.7.34 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.

### 13.3 FlexCAN eDMA Driver

#### 13.3.1 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.*

#### 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.*

#### 13.3.2 Data Structure Documentation

##### 13.3.2.1 struct \_flexcan\_edma\_handle

#### Data Fields

- `flexcan_edma_transfer_callback_t` `callback`  
*Callback function.*
- void \* `userData`  
*FlexCAN callback function parameter.*
- `edma_handle_t` \* `rxFifoEdmaHandle`  
*The EDMA Rx FIFO channel used.*
- volatile uint8\_t `rxFifoState`  
*Rx FIFO transfer state.*

### 13.3.2.1.0.23 Field Documentation

13.3.2.1.0.23.1 `flexcan_edma_transfer_callback_t flexcan_edma_handle_t::callback`

13.3.2.1.0.23.2 `void* flexcan_edma_handle_t::userData`

13.3.2.1.0.23.3 `edma_handle_t* flexcan_edma_handle_t::rxFifoEdmaHandle`

13.3.2.1.0.23.4 `volatile uint8_t flexcan_edma_handle_t::rxFifoState`

### 13.3.3 Typedef Documentation

13.3.3.1 `typedef void(* flexcan_edma_transfer_callback_t)(CAN_Type *base,  
flexcan_edma_handle_t *handle, status_t status, void *userData)`

### 13.3.4 Function Documentation

13.3.4.1 `void FLEXCAN_TransferCreateHandleEDMA ( CAN_Type * base,  
flexcan_edma_handle_t * handle, flexcan_edma_transfer_callback_t callback,  
void * userData, edma_handle_t * rxFifoEdmaHandle )`

## FlexCAN eDMA Driver

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>rxFifoEdmaHandle</i>	User-requested DMA handle for Rx FIFO DMA transfer.

### 13.3.4.2 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.

### 13.3.4.3 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 14

## GPT: General Purpose Timer

### 14.1 Overview

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

### 14.2 Function groups

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

#### 14.2.1 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.

### 14.3 Typical use case

#### 14.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second.

```
int main(void)
{
    uint32_t gptFreq;
    gpt_config_t gptConfig;

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    GPT_GetDefaultConfig(&gptConfig);

    /* Initialize GPT module */
    GPT_Init(EXAMPLE_GPT, &gptConfig);

    /* Divide GPT clock source frequency by 3 inside GPT module */
    GPT_SetClockDivider(EXAMPLE_GPT, 3);

    /* Get GPT clock frequency */
    gptFreq = EXAMPLE_GPT_CLK_FREQ;

    /* GPT frequency is divided by 3 inside module */
    gptFreq /= 3;

    /* Set both GPT modules to 1 second duration */
```

## Typical use case

```
GPT_SetOutputCompareValue(EXAMPLE_GPT,  
    kGPT_OutputCompare_Channel1, gptFreq);  
  
/* Enable GPT Output Compare1 interrupt */  
GPT_EnableInterrupts(EXAMPLE_GPT,  
    kGPT_OutputCompare1InterruptEnable);  
  
/* Enable at the Interrupt */  
EnableIRQ(GPT IRQ_ID);  
  
/* Start Timer */  
PRINTF("\r\nStarting GPT timer ...");  
GPT_StartTimer(EXAMPLE_GPT);  
  
while (true)  
{  
    /* Check whether occur interrupt and toggle LED */  
    if (true == gptIsrFlag)  
    {  
        PRINTF("\r\n GPT interrupt is occurred !");  
        gptIsrFlag = false;  
    }  
    else  
    {  
        __WFI();  
    }  
}  
}
```

## 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\_Pерiph = 1U,  
 kGPT\_ClockSource\_HighFreq = 2U,  
 kGPT\_ClockSource\_Ext = 3U,  
 kGPT\_ClockSource\_LowFreq = 4U,  
 kGPT\_ClockSource\_Osc = 5U }  
*List of clock sources.*
- enum [gpt\\_input\\_capture\\_channel\\_t](#) {  
 kGPT\_InputCapture\_Channel1 = 0U,  
 kGPT\_InputCapture\_Channel2 = 1U }  
*List of input capture channel number.*
- enum [gpt\\_input\\_operation\\_mode\\_t](#) {  
 kGPT\_InputOperation\_Disabled = 0U,  
 kGPT\_InputOperation\_RiseEdge = 1U,  
 kGPT\_InputOperation\_FallEdge = 2U,  
 kGPT\_InputOperation\_BothEdge = 3U }  
*List of input capture operation mode.*
- enum [gpt\\_output\\_compare\\_channel\\_t](#) {

```
kGPT_OutputCompare_Channel1 = 0U,
kGPT_OutputCompare_Channel2 = 1U,
kGPT_OutputCompare_Channel3 = 2U }
```

*List of output compare channel number.*

- enum `gpt_output_operation_mode_t` {
 kGPT\_OutputOperation\_Disconnected = 0U,
 kGPT\_OutputOperation\_Toggle = 1U,
 kGPT\_OutputOperation\_Clear = 2U,
 kGPT\_OutputOperation\_Set = 3U,
 kGPT\_OutputOperation\_Activelow = 4U }

*List of output compare operation mode.*

- enum `gpt_interrupt_enable_t` {
 kGPT\_OutputCompare1InterruptEnable = GPT\_IR\_OF1IE\_MASK,
 kGPT\_OutputCompare2InterruptEnable = GPT\_IR\_OF2IE\_MASK,
 kGPT\_OutputCompare3InterruptEnable = GPT\_IR\_OF3IE\_MASK,
 kGPT\_InputCapture1InterruptEnable = GPT\_IR\_IF1IE\_MASK,
 kGPT\_InputCapture2InterruptEnable = GPT\_IR\_IF2IE\_MASK,
 kGPT\_RollOverFlagInterruptEnable = GPT\_IR\_ROVIE\_MASK }

*List of GPT interrupts.*

- enum `gpt_status_flag_t` {
 kGPT\_OutputCompare1Flag = GPT\_SR\_OF1\_MASK,
 kGPT\_OutputCompare2Flag = GPT\_SR\_OF2\_MASK,
 kGPT\_OutputCompare3Flag = GPT\_SR\_OF3\_MASK,
 kGPT\_InputCapture1Flag = GPT\_SR\_IF1\_MASK,
 kGPT\_InputCapture2Flag = GPT\_SR\_IF2\_MASK,
 kGPT\_RollOverFlag = GPT\_SR\_ROV\_MASK }

*Status flag.*

## Driver version

- #define `FSL_GPT_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 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.*

## Typical use case

### 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.*
- 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.*

## 14.4 Data Structure Documentation

### 14.4.1 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.*

## Enumeration Type Documentation

### 14.4.1.0.0.24 Field Documentation

14.4.1.0.0.24.1 `gpt_clock_source_t gpt_config_t::clockSource`

14.4.1.0.0.24.2 `uint32_t gpt_config_t::divider`

14.4.1.0.0.24.3 `bool gpt_config_t::enableFreeRun`

14.4.1.0.0.24.4 `bool gpt_config_t::enableRunInWait`

14.4.1.0.0.24.5 `bool gpt_config_t::enableRunInStop`

14.4.1.0.0.24.6 `bool gpt_config_t::enableRunInDoze`

14.4.1.0.0.24.7 `bool gpt_config_t::enableRunInDbg`

14.4.1.0.0.24.8 `bool gpt_config_t::enableMode`

## 14.5 Enumeration Type Documentation

### 14.5.1 enum gpt\_clock\_source\_t

Note

Actual number of clock sources is SoC dependent

Enumerator

*kGPT\_ClockSource\_Off* GPT Clock Source Off.

*kGPT\_ClockSource\_Pерiph* 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.

### 14.5.2 enum gpt\_input\_capture\_channel\_t

Enumerator

*kGPT\_InputCapture\_Channel1* GPT Input Capture Channel1.

*kGPT\_InputCapture\_Channel2* GPT Input Capture Channel2.

### 14.5.3 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.

#### 14.5.4 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.

#### 14.5.5 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.

#### 14.5.6 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.

#### 14.5.7 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.

## Function Documentation

*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.

## 14.6 Function Documentation

### 14.6.1 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.

### 14.6.2 void GPT\_Deinit ( **GPT\_Type** \* *base* )

Parameters

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

### 14.6.3 void GPT\_GetDefaultConfig ( **gpt\_config\_t** \* *config* )

The default values are:

```
* config->clockSource = kGPT_ClockSource_Pерiph;
* config->divider = 1U;
* config->enableRunInStop = true;
* config->enableRunInWait = true;
* config->enableRunInDoze = false;
* config->enableRunInDbg = false;
* config->enableFreeRun = true;
* config->enableMode = true;
*
```

Parameters

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

### 14.6.4 static void GPT\_SoftwareReset ( **GPT\_Type** \* *base* ) [inline], [static]

Parameters

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

#### 14.6.5 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).

#### 14.6.6 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).

#### 14.6.7 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).

#### 14.6.8 static **uint32\_t** GPT\_GetClockDivider ( **GPT\_Type** \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

Returns

clock divider in GPT module (1-4096).

**14.6.9 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).

**14.6.10 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).

**14.6.11 static void GPT\_StartTimer ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

**14.6.12 static void GPT\_StopTimer ( GPT\_Type \* *base* ) [inline], [static]**

Parameters

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

**14.6.13 static uint32\_t GPT\_GetCurrentTimerCount ( GPT\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

Current GPT counter value.

**14.6.14 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).

**14.6.15 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.
-------------	------------------------------

## Function Documentation

<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).

### 14.6.16 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.

### 14.6.17 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).

### 14.6.18 static **gpt\_output\_operation\_mode\_t** GPT\_GetOutputOperationMode ( **GPT\_Type** \* *base*, **gpt\_output\_compare\_channel\_t** *channel* ) [inline], [static]

Parameters

<i>base</i>	GPT peripheral base address.
<i>channel</i>	GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration).

Returns

GPT output operation mode (see [gpt\\_output\\_operation\\_mode\\_t](#) typedef enumeration).

**14.6.19 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.

**14.6.20 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.

**14.6.21 static void GPT\_ForceOutput ( 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).

**14.6.22 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>

**14.6.23 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>

**14.6.24 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](#)

14.6.25 **static uint32\_t GPT\_GetStatusFlags ( GPT\_Type \* *base*, gpt\_status\_flag\_t *flags* ) [inline], [static]**

## Function Documentation

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.

**14.6.26 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 15

## GPC: General Power Controller Driver

### 15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Power Controller (GPC) module of MCUXpresso SDK devices.

API functions are provided to configure the system about working in dedicated power mode. There are mainly about enabling the power for memory, enabling the wakeup sources for STOP modes and power up/down operations for various peripherals.

### Functions

- static void [GPC\\_AllowIRQs](#) (GPC\_Type \*base)  
*Allow all the IRQ/Events within the charge of GPC.*
- static void [GPC\\_DisallowIRQs](#) (GPC\_Type \*base)  
*Disallow all the IRQ/Events within the charge of GPC.*
- void [GPC\\_EnableIRQ](#) (GPC\_Type \*base, uint32\_t irqId)  
*Enable the IRQ.*
- void [GPC\\_DisableIRQ](#) (GPC\_Type \*base, uint32\_t irqId)  
*Disable the IRQ.*
- bool [GPC\\_GetIRQStatusFlag](#) (GPC\_Type \*base, uint32\_t irqId)  
*Get the IRQ/Event flag.*
- static void [GPC\\_RequestL2CachePowerDown](#) (GPC\_Type \*base, bool enable)  
*L2 Cache Power Gate Enable.*
- static void [GPC\\_RequestVADCPowerDown](#) (GPC\_Type \*base, bool enable)  
*VADC power down.*
- static bool [GPC\\_GetVADCPowerDownFlag](#) (GPC\_Type \*base)  
*Checks if the VADC is power off.*
- static bool [GPC\\_HasDVFS0ChangeRequest](#) (GPC\_Type \*base)  
*Checks if the DVFS0 is requesting for frequency/voltage update.*
- static void [GPC\\_RequestDisplayPowerOn](#) (GPC\_Type \*base, bool enable)  
*Requests the display power switch sequence.*
- static void [GPC\\_RequestMEGAPowerOn](#) (GPC\_Type \*base, bool enable)  
*Requests the MEGA power switch sequence.*

### Driver version

- #define [FSL\\_GPC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*GPC driver version 2.0.0.*

### 15.2 Macro Definition Documentation

#### 15.2.1 #define FSL\_GPC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## Function Documentation

### 15.3 Function Documentation

15.3.1 **static void GPC\_AllowIRQs ( GPC\_Type \* *base* ) [inline], [static]**

Parameters

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

### 15.3.2 static void GPC\_DisallowIRQs ( GPC\_Type \* *base* ) [inline], [static]

Parameters

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

### 15.3.3 void GPC\_EnableIRQ ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

<i>base</i>	GPC peripheral base address.
<i>irqId</i>	ID number of IRQ to be enabled, available range is 32-159.

### 15.3.4 void GPC\_DisableIRQ ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

<i>base</i>	GPC peripheral base address.
<i>irqId</i>	ID number of IRQ to be disabled, available range is 32-159.

### 15.3.5 bool GPC\_GetIRQStatusFlag ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

<i>base</i>	GPC peripheral base address.
<i>irqId</i>	ID number of IRQ to be enabled, available range is 32-159.

Returns

Indicated IRQ/Event is asserted or not.

## Function Documentation

### 15.3.6 static void GPC\_RequestL2CachePowerDown ( *GPC\_Type* \* *base*, *bool enable* ) [inline], [static]

This function configures the L2 cache if it will keep power when in low power mode. When the L2 cache power is OFF, L2 cache will be power down once when CPU core is power down and will be hardware invalidated automatically when CPU core is re-power up. When the L2 cache power is ON, L2 cache will keep power on even if CPU core is power down and will not be hardware invalidated. When CPU core is re-power up, the default setting is OFF.

Parameters

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

### 15.3.7 static void GPC\_RequestVADCPowerDown ( *GPC\_Type* \* *base*, *bool enable* ) [inline], [static]

This function requests the VADC power down.

Parameters

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

### 15.3.8 static bool GPC\_GetVADCPowerDownFlag ( *GPC\_Type* \* *base* ) [inline], [static]

Parameters

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

Returns

Whether the VADC is power off or not.

### 15.3.9 static bool GPC\_HasDVFS0ChangeRequest ( *GPC\_Type* \* *base* ) [inline], [static]

Parameters

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

Returns

Whether the DVFS0 is requesting for frequency/voltage update.

### 15.3.10 static void GPC\_RequestDisplayPowerOn ( GPC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	GPC peripheral base address.
<i>enable</i>	Enable the power on sequence, or the power down sequence.

### 15.3.11 static void GPC\_RequestMEGAPowerOn ( GPC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	GPC peripheral base address.
<i>enable</i>	Enable the power on sequence, or the power down sequence.

## Function Documentation

# **Chapter 16**

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

### **16.1 Overview**

#### **Modules**

- [GPIO Driver](#)

### 16.2 GPIO Driver

#### 16.2.1 Overview

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

#### 16.2.2 Typical use case

##### 16.2.2.1 Input Operation

```
/* Input pin configuration */
EnableIRQ(EXAMPLE_SW_IRQ);
gpio_pin_config_t sw_config = {
    kGPIO_DigitalInput, 0,
    kGPIO_IntRisingEdge,
};

/* Sets the input pin configuration */
GPIO_PinInit(EXAMPLE_SW_GPIO, EXAMPLE_SW_GPIO_PIN, &sw_config);
```

## 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, 0))  
*GPIO driver version 2.0.0.*

## 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\_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\_SetPinsOutput** (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void **GPIO\_ClearPinsOutput** (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static 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\_ReadPadStatus** (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*

## Interrupts and flags management functions

- void **GPIO\_SetPinInterruptConfig** (GPIO\_Type \*base, uint32\_t pin, **gpio\_interrupt\_mode\_t** pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void **GPIO\_EnableInterrupts** (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void **GPIO\_DisableInterrupts** (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static uint32\_t **GPIO\_GetPinsInterruptFlags** (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*
- static void **GPIO\_ClearPinsInterruptFlags** (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*

### 16.2.3 Data Structure Documentation

#### 16.2.3.1 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 Driver

- `gpio_interrupt_mode_t interruptMode`  
*Specifies the pin interrupt mode, a value of `gpio_interrupt_mode_t`.*

### 16.2.3.1.0.25 Field Documentation

**16.2.3.1.0.25.1 `gpio_pin_direction_t gpio_pin_config_t::direction`**

**16.2.3.1.0.25.2 `gpio_interrupt_mode_t gpio_pin_config_t::interruptMode`**

### 16.2.4 Macro Definition Documentation

**16.2.4.1 `#define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`**

### 16.2.5 Enumeration Type Documentation

#### 16.2.5.1 `enum gpio_pin_direction_t`

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.

*kGPIO\_DigitalOutput* Set current pin as digital output.

#### 16.2.5.2 `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.

### 16.2.6 Function Documentation

**16.2.6.1 `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.

#### 16.2.6.2 void GPIO\_WritePinOutput ( **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>

#### 16.2.6.3 static void GPIO\_SetPinsOutput ( **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

#### 16.2.6.4 static void GPIO\_ClearPinsOutput ( **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

#### 16.2.6.5 static **uint32\_t** GPIO\_ReadPinInput ( **GPIO\_Type** \* *base*, **uint32\_t** *pin* ) [**inline**], [**static**]

## GPIO Driver

Parameters

<i>base</i>	GPIO base pointer.
<i>pin</i>	GPIO port pin number.

Return values

<i>GPIO</i>	port input value.
-------------	-------------------

**16.2.6.6 static uint8\_t GPIO\_ReadPadStatus ( **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.
-------------	-----------------------

**16.2.6.7 void GPIO\_SetPinInterruptConfig ( **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.

**16.2.6.8 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.

**16.2.6.9 static void GPIO\_DisableInterrupts ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

<i>base</i>	GPIO base pointer.
<i>mask</i>	GPIO pin number macro.

**16.2.6.10 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.
----------------	----------------------------

**16.2.6.11 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 17**

## **I2C: Inter-Integrated Circuit Driver**

### **17.1 Overview**

#### **Modules**

- I2C Driver
- I2C FreeRTOS Driver

### 17.2 I2C Driver

#### 17.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs target the low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires knowing the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs target the high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions [I2C\\_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

#### 17.2.2 Typical use case

##### 17.2.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Gets the default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Initializes the I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Sends a start and a slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
    kI2C_Write/kI2C_Read);

/* Waits for the sent out address. */
while(!((status = I2C_MasterGetStatusFlags(EXAMPLE_I2C_MASTER_BASEADDR)) &
        kI2C_IntPendingFlag))
{
}

if(status & kI2C_ReceiveNakFlag)
{
    return kStatus_I2C_Nak;
}

result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE,
    kI2C_TransferDefaultFlag);
```

```
return result;
```

### 17.2.2.2 Master Operation in interrupt transactional method

```
i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *
    userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Gets a default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Initializes the I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
    i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &
    masterXfer);

/* Waits for a transfer to be completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

### 17.2.2.3 Slave Operation in functional method

```
i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*A default configuration 7-bit
    addressing mode*/
slaveConfig.slaveAddr = 7-bit address;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

/* Waits for an address match. */
while(!((status = I2C_SlaveGetStatusFlags(EXAMPLE_I2C_SLAVE_BASEADDR)) &
    kI2C_AddressMatchFlag))
{
```

## I2C Driver

```
}

/* A slave transmits; master is reading from the slave. */
if (status & kI2C_TransferDirectionFlag)
{
    result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}
else
{
    I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}

return result;
```

### 17.2.2.4 Slave Operation in interrupt transactional method

```
i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
    userData)
{
    switch (xfer->event)
    {
        /* Transmit request */
        case kI2C_SlaveTransmitEvent:
            /* Update information for transmit process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /* Receives request */
        case kI2C_SlaveReceiveEvent:
            /* Update information for received process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /* Transfer is done */
        case kI2C_SlaveCompletionEvent:
            g_SlaveCompletionFlag = true;
            break;

        default:
            g_SlaveCompletionFlag = true;
            break;
    }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*A default configuration 7-bit
    addressing mode*/
slaveConfig.slaveAddr = 7-bit address;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
    i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
    kI2C_SlaveCompletionEvent);

/* Waits for a transfer to be completed. */
while (!g_SlaveCompletionFlag)
{
```

```
}

g_SlaveCompletionFlag = false;
```

## Data Structures

- struct `i2c_master_config_t`  
*I2C master user configuration.* [More...](#)
- struct `i2c_master_transfer_t`  
*I2C master transfer structure.* [More...](#)
- struct `i2c_master_handle_t`  
*I2C master handle structure.* [More...](#)
- struct `i2c_slave_config_t`  
*I2C slave user configuration.* [More...](#)
- struct `i2c_slave_transfer_t`  
*I2C slave transfer structure.* [More...](#)
- struct `i2c_slave_handle_t`  
*I2C slave handle structure.* [More...](#)

## Typedefs

- typedef void(\* `i2c_master_transfer_callback_t`)  
(I2C\_Type \*base, `i2c_master_handle_t` \*handle,  
status\_t status, void \*userData)  
*I2C master transfer callback typedef.*
- typedef void(\* `i2c_slave_transfer_callback_t`)  
(I2C\_Type \*base, `i2c_slave_transfer_t` \*xfer, void  
\*userData)  
*I2C slave transfer callback typedef.*

## Enumerations

- enum `_i2c_status` {
   
kStatus\_I2C\_Busy = MAKE\_STATUS(kStatusGroup\_I2C, 0),
   
kStatus\_I2C\_Idle = MAKE\_STATUS(kStatusGroup\_I2C, 1),
   
kStatus\_I2C\_Nak = MAKE\_STATUS(kStatusGroup\_I2C, 2),
   
kStatus\_I2C\_ArbitrationLost = MAKE\_STATUS(kStatusGroup\_I2C, 3),
   
kStatus\_I2C\_Timeout = MAKE\_STATUS(kStatusGroup\_I2C, 4),
   
kStatus\_I2C\_Addr\_Nak = MAKE\_STATUS(kStatusGroup\_I2C, 5) }
   
*I2C status return codes.*
- enum `_i2c_flags` {
   
kI2C\_ReceiveNakFlag = I2C\_I2SR\_RXAK\_MASK,
   
kI2C\_IntPendingFlag = I2C\_I2SR\_IIF\_MASK,
   
kI2C\_TransferDirectionFlag = I2C\_I2SR\_SRW\_MASK,
   
kI2C\_ArbitrationLostFlag = I2C\_I2SR\_IAL\_MASK,
   
kI2C\_BusBusyFlag = I2C\_I2SR\_IBB\_MASK,
   
kI2C\_AddressMatchFlag = I2C\_I2SR\_IAAS\_MASK,
   
kI2C\_TransferCompleteFlag = I2C\_I2SR\_ICF\_MASK }

## I2C Driver

- enum `_i2c_interrupt_enable` { `kI2C_GlobalInterruptEnable` = I2C\_I2CR\_IIEN\_MASK }
- I2C feature interrupt source.*
- enum `i2c_direction_t` {
  - `kI2C_Write` = 0x0U,
  - `kI2C_Read` = 0x1U }
- The direction of master and slave transfers.*
- enum `_i2c_master_transfer_flags` {
  - `kI2C_TransferDefaultFlag` = 0x0U,
  - `kI2C_TransferNoStartFlag` = 0x1U,
  - `kI2C_TransferRepeatedStartFlag` = 0x2U,
  - `kI2C_TransferNoStopFlag` = 0x4U }
- I2C transfer control flag.*
- enum `i2c_slave_transfer_event_t` {
  - `kI2C_SlaveAddressMatchEvent` = 0x01U,
  - `kI2C_SlaveTransmitEvent` = 0x02U,
  - `kI2C_SlaveReceiveEvent` = 0x04U,
  - `kI2C_SlaveTransmitAckEvent` = 0x08U,
  - `kI2C_SlaveCompletionEvent` = 0x20U,
  - `kI2C_SlaveAllEvents` }
- Set of events sent to the callback for nonblocking slave transfers.*

## Driver version

- #define `FSL_I2C_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*I2C driver version 2.0.0.*

## Initialization and deinitialization

- void `I2C_MasterInit` (I2C\_Type \*base, const `i2c_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the I2C peripheral.*
- void `I2C_MasterDeinit` (I2C\_Type \*base)  
*De-initializes the I2C master peripheral.*
- void `I2C_MasterGetDefaultConfig` (`i2c_master_config_t` \*masterConfig)  
*Sets the I2C master configuration structure to default values.*
- void `I2C_SlaveInit` (I2C\_Type \*base, const `i2c_slave_config_t` \*slaveConfig)  
*Initializes the I2C peripheral.*
- void `I2C_SlaveDeinit` (I2C\_Type \*base)  
*De-initializes the I2C slave peripheral.*
- void `I2C_SlaveGetDefaultConfig` (`i2c_slave_config_t` \*slaveConfig)  
*Sets the I2C slave configuration structure to default values.*
- static void `I2C_Enable` (I2C\_Type \*base, bool enable)  
*Enables or disables the I2C peripheral operation.*

## Status

- static uint32\_t **I2C\_MasterGetStatusFlags** (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static void **I2C\_MasterClearStatusFlags** (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*
- static uint32\_t **I2C\_SlaveGetStatusFlags** (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static void **I2C\_SlaveClearStatusFlags** (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*

## Interrupts

- void **I2C\_EnableInterrupts** (I2C\_Type \*base, uint32\_t mask)  
*Enables I2C interrupt requests.*
- void **I2C\_DisableInterrupts** (I2C\_Type \*base, uint32\_t mask)  
*Disables I2C interrupt requests.*

## Bus Operations

- void **I2C\_MasterSetBaudRate** (I2C\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the I2C master transfer baud rate.*
- status\_t **I2C\_MasterStart** (I2C\_Type \*base, uint8\_t address, i2c\_direction\_t direction)  
*Sends a START on the I2C bus.*
- status\_t **I2C\_MasterStop** (I2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- status\_t **I2C\_MasterRepeatedStart** (I2C\_Type \*base, uint8\_t address, i2c\_direction\_t direction)  
*Sends a REPEATED START on the I2C bus.*
- status\_t **I2C\_MasterWriteBlocking** (I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize, uint32\_t flags)  
*Performs a polling send transaction on the I2C bus.*
- status\_t **I2C\_MasterReadBlocking** (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize, uint32\_t flags)  
*Performs a polling receive transaction on the I2C bus.*
- status\_t **I2C\_SlaveWriteBlocking** (I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize)  
*Performs a polling send transaction on the I2C bus.*
- void **I2C\_SlaveReadBlocking** (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize)  
*Performs a polling receive transaction on the I2C bus.*
- status\_t **I2C\_MasterTransferBlocking** (I2C\_Type \*base, i2c\_master\_transfer\_t \*xfer)  
*Performs a master polling transfer on the I2C bus.*

## Transactional

- void **I2C\_MasterTransferCreateHandle** (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, i2c\_master\_transfer\_callback\_t callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- status\_t **I2C\_MasterTransferNonBlocking** (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, i2c\_master\_transfer\_t \*xfer)

## I2C Driver

*Performs a master interrupt non-blocking transfer on the I2C bus.*

- status\_t [I2C\\_MasterTransferGetCount](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- void [I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
*Master interrupt handler.*
- void [I2C\\_SlaveTransferCreateHandle](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, [i2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- status\_t [I2C\\_SlaveTransferNonBlocking](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- void [I2C\\_SlaveTransferAbort](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)  
*Aborts the slave transfer.*
- status\_t [I2C\\_SlaveTransferGetCount](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.*
- void [I2C\\_SlaveTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
*Slave interrupt handler.*

### 17.2.3 Data Structure Documentation

#### 17.2.3.1 struct i2c\_master\_config\_t

##### Data Fields

- bool [enableMaster](#)  
*Enables the I2C peripheral at initialization time.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud rate configuration of I2C peripheral.*

##### 17.2.3.1.0.26 Field Documentation

###### 17.2.3.1.0.26.1 bool i2c\_master\_config\_t::enableMaster

###### 17.2.3.1.0.26.2 uint32\_t i2c\_master\_config\_t::baudRate\_Bps

#### 17.2.3.2 struct i2c\_master\_transfer\_t

##### Data Fields

- uint32\_t [flags](#)  
*A transfer flag which controls the transfer.*
- uint8\_t [slaveAddress](#)  
*7-bit slave address.*
- [i2c\\_direction\\_t](#) [direction](#)  
*A transfer direction, read or write.*
- uint32\_t [subaddress](#)

- **uint8\_t subaddressSize**  
*A sub address.*
- **uint8\_t \*volatile data**  
*A transfer buffer.*
- **volatile size\_t dataSize**  
*A transfer size.*

### 17.2.3.2.0.27 Field Documentation

**17.2.3.2.0.27.1 uint32\_t i2c\_master\_transfer\_t::flags**

**17.2.3.2.0.27.2 uint8\_t i2c\_master\_transfer\_t::slaveAddress**

**17.2.3.2.0.27.3 i2c\_direction\_t i2c\_master\_transfer\_t::direction**

**17.2.3.2.0.27.4 uint32\_t i2c\_master\_transfer\_t::subaddress**

Transferred MSB first.

**17.2.3.2.0.27.5 uint8\_t i2c\_master\_transfer\_t::subaddressSize**

**17.2.3.2.0.27.6 uint8\_t\* volatile i2c\_master\_transfer\_t::data**

**17.2.3.2.0.27.7 volatile size\_t i2c\_master\_transfer\_t::dataSize**

### 17.2.3.3 struct \_i2c\_master\_handle

I2C master handle typedef.

## Data Fields

- **i2c\_master\_transfer\_t transfer**  
*I2C master transfer copy.*
- **size\_t transferSize**  
*Total bytes to be transferred.*
- **uint8\_t state**  
*A transfer state maintained during transfer.*
- **i2c\_master\_transfer\_callback\_t completionCallback**  
*A callback function called when the transfer is finished.*
- **void \*userData**  
*A callback parameter passed to the callback function.*

## I2C Driver

### 17.2.3.3.0.28 Field Documentation

17.2.3.3.0.28.1 `i2c_master_transfer_t i2c_master_handle_t::transfer`

17.2.3.3.0.28.2 `size_t i2c_master_handle_t::transferSize`

17.2.3.3.0.28.3 `uint8_t i2c_master_handle_t::state`

17.2.3.3.0.28.4 `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`

17.2.3.3.0.28.5 `void* i2c_master_handle_t::userData`

### 17.2.3.4 struct `i2c_slave_config_t`

#### Data Fields

- `bool enableSlave`  
*Enables the I2C peripheral at initialization time.*
- `uint16_t slaveAddress`  
*A slave address configuration.*

### 17.2.3.4.0.29 Field Documentation

17.2.3.4.0.29.1 `bool i2c_slave_config_t::enableSlave`

17.2.3.4.0.29.2 `uint16_t i2c_slave_config_t::slaveAddress`

### 17.2.3.5 struct `i2c_slave_transfer_t`

#### Data Fields

- `i2c_slave_transfer_event_t event`  
*A reason that the callback is invoked.*
- `uint8_t *volatile data`  
*A transfer buffer.*
- `volatile size_t dataSize`  
*A transfer size.*
- `status_t completionStatus`  
*Success or error code describing how the transfer completed.*
- `size_t transferredCount`  
*A number of bytes actually transferred since the start or since the last repeated start.*

### 17.2.3.5.0.30 Field Documentation

17.2.3.5.0.30.1 `i2c_slave_transfer_event_t i2c_slave_transfer_t::event`

17.2.3.5.0.30.2 `uint8_t* volatile i2c_slave_transfer_t::data`

17.2.3.5.0.30.3 `volatile size_t i2c_slave_transfer_t::dataSize`

17.2.3.5.0.30.4 `status_t i2c_slave_transfer_t::completionStatus`

Only applies for [kI2C\\_SlaveCompletionEvent](#).

17.2.3.5.0.30.5 `size_t i2c_slave_transfer_t::transferredCount`

### 17.2.3.6 struct \_i2c\_slave\_handle

I2C slave handle typedef.

#### Data Fields

- `volatile uint8_t state`  
*A transfer state maintained during transfer.*
- `i2c_slave_transfer_t transfer`  
*I2C slave transfer copy.*
- `uint32_t eventMask`  
*A mask of enabled events.*
- `i2c_slave_transfer_callback_t callback`  
*A callback function called at the transfer event.*
- `void * userData`  
*A callback parameter passed to the callback.*

## I2C Driver

### 17.2.3.6.0.31 Field Documentation

17.2.3.6.0.31.1 `volatile uint8_t i2c_slave_handle_t::state`

17.2.3.6.0.31.2 `i2c_slave_transfer_t i2c_slave_handle_t::transfer`

17.2.3.6.0.31.3 `uint32_t i2c_slave_handle_t::eventMask`

17.2.3.6.0.31.4 `i2c_slave_transfer_callback_t i2c_slave_handle_t::callback`

17.2.3.6.0.31.5 `void* i2c_slave_handle_t::userData`

### 17.2.4 Macro Definition Documentation

17.2.4.1 `#define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

### 17.2.5 Typedef Documentation

17.2.5.1 `typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *userData)`

17.2.5.2 `typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, i2c_slave_transfer_t *xfer, void *userData)`

### 17.2.6 Enumeration Type Documentation

#### 17.2.6.1 enum \_i2c\_status

Enumerator

*kStatus\_I2C\_Busy* I2C is busy with current transfer.

*kStatus\_I2C\_Idle* Bus is Idle.

*kStatus\_I2C\_Nak* NAK received during transfer.

*kStatus\_I2C\_ArbitrationLost* Arbitration lost during transfer.

*kStatus\_I2C\_Timeout* Wait event timeout.

*kStatus\_I2C\_Addr\_Nak* NAK received during the address probe.

#### 17.2.6.2 enum \_i2c\_flags

The following status register flags can be cleared:

- `kI2C_ArbitrationLostFlag`
- `kI2C_IntPendingFlag`

## Note

These enumerations are meant to be OR'd together to form a bit mask.

## Enumerator

- kI2C\_ReceiveNakFlag*** I2C receive NAK flag.
- kI2C\_IntPendingFlag*** I2C interrupt pending flag.
- kI2C\_TransferDirectionFlag*** I2C transfer direction flag.
- kI2C\_ArbitrationLostFlag*** I2C arbitration lost flag.
- kI2C\_BusBusyFlag*** I2C bus busy flag.
- kI2C\_AddressMatchFlag*** I2C address match flag.
- kI2C\_TransferCompleteFlag*** I2C transfer complete flag.

**17.2.6.3 enum \_i2c\_interrupt\_enable**

## Enumerator

- kI2C\_GlobalInterruptEnable*** I2C global interrupt.

**17.2.6.4 enum i2c\_direction\_t**

## Enumerator

- kI2C\_Write*** Master transmits to the slave.
- kI2C\_Read*** Master receives from the slave.

**17.2.6.5 enum \_i2c\_master\_transfer\_flags**

## Enumerator

- kI2C\_TransferDefaultFlag*** A transfer starts with a start signal, stops with a stop signal.
- kI2C\_TransferNoStartFlag*** A transfer starts without a start signal.
- kI2C\_TransferRepeatedStartFlag*** A transfer starts with a repeated start signal.
- kI2C\_TransferNoStopFlag*** A transfer ends without a stop signal.

**17.2.6.6 enum i2c\_slave\_transfer\_event\_t**

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to [I2C\\_SlaveTransferNonBlocking\(\)](#) to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

## I2C Driver

### Note

These enumerations are meant to be OR'd together to form a bit mask of events.

### Enumerator

***kI2C\_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.

***kI2C\_SlaveTransmitEvent*** A callback is requested to provide data to transmit (slave-transmitter role).

***kI2C\_SlaveReceiveEvent*** A callback is requested to provide a buffer in which to place received data (slave-receiver role).

***kI2C\_SlaveTransmitAckEvent*** A callback needs to either transmit an ACK or NACK.

***kI2C\_SlaveCompletionEvent*** A stop was detected or finished transfer, completing the transfer.

***kI2C\_SlaveAllEvents*** A bit mask of all available events.

### 17.2.7 Function Documentation

#### 17.2.7.1 void I2C\_MasterInit ( ***I2C\_Type*** \* *base*, ***const i2c\_master\_config\_t*** \* *masterConfig*, ***uint32\_t srcClock\_Hz*** )

Call this API to ungate the I2C clock and configure the I2C with master configuration.

### Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the [I2C\\_MasterGetDefaultConfig\(\)](#). After calling this API, the master is ready to transfer. This is an example.

```
* i2c_master_config_t config = {  
*   .enableMaster = true,  
*   .baudRate_Bps = 100000  
* };  
* I2C_MasterInit(I2C0, &config, 12000000U);  
*
```

### Parameters

<i>base</i>	I2C base pointer
<i>masterConfig</i>	A pointer to the master configuration structure
<i>srcClock_Hz</i>	I2C peripheral clock frequency in Hz

#### 17.2.7.2 void I2C\_MasterDeinit ( ***I2C\_Type*** \* *base* )

Call this API to gate the I2C clock. The I2C master module can't work unless the I2C\_MasterInit is called.

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

### 17.2.7.3 void I2C\_MasterGetDefaultConfig ( i2c\_master\_config\_t \* *masterConfig* )

The purpose of this API is to get the configuration structure initialized for use in the [I2C\\_MasterInit\(\)](#). Use the initialized structure unchanged in the [I2C\\_MasterInit\(\)](#) or modify the structure before calling the [I2C\\_MasterInit\(\)](#). This is an example.

```
* i2c_master_config_t config;
* I2C_MasterGetDefaultConfig(&config);
*
```

Parameters

<i>masterConfig</i>	A pointer to the master configuration structure.
---------------------	--

### 17.2.7.4 void I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig* )

Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by [I2C\\_SlaveGetDefaultConfig\(\)](#) or it can be custom filled by the user. This is an example.

```
* i2c_slave_config_t config = {
* .enableSlave = true,
* .slaveAddress = 0x1DU,
* };
* I2C_SlaveInit(I2C0, &config);
*
```

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

## I2C Driver

<i>slaveConfig</i>	A pointer to the slave configuration structure
<i>srcClock_Hz</i>	I2C peripheral clock frequency in Hz

### 17.2.7.5 void I2C\_SlaveDeinit ( I2C\_Type \* *base* )

Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C\_SlaveInit is called to enable the clock.

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

### 17.2.7.6 void I2C\_SlaveGetDefaultConfig ( i2c\_slave\_config\_t \* *slaveConfig* )

The purpose of this API is to get the configuration structure initialized for use in the [I2C\\_SlaveInit\(\)](#). Modify fields of the structure before calling the [I2C\\_SlaveInit\(\)](#). This is an example.

```
* i2c_slave_config_t config;
* I2C_SlaveGetDefaultConfig(&config);
*
```

Parameters

<i>slaveConfig</i>	A pointer to the slave configuration structure.
--------------------	---

### 17.2.7.7 static void I2C\_Enable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	I2C base pointer
<i>enable</i>	Pass true to enable and false to disable the module.

### 17.2.7.8 static uint32\_t I2C\_MasterGetStatusFlags ( I2C\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

#### 17.2.7.9 static void I2C\_MasterClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag.

Parameters

<i>base</i>	I2C base pointer
<i>statusMask</i>	<p>The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> <li>• kI2C_ArbitrationLostFlag</li> <li>• kI2C_IntPendingFlagFlag</li> </ul>

#### 17.2.7.10 static uint32\_t I2C\_SlaveGetStatusFlags ( I2C\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	I2C base pointer
-------------	------------------

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

#### 17.2.7.11 static void I2C\_SlaveClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag

## I2C Driver

Parameters

<i>base</i>	I2C base pointer
<i>statusMask</i>	The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• kI2C_IntPendingFlagFlag</li></ul>

### 17.2.7.12 void I2C\_EnableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	I2C base pointer
<i>mask</i>	interrupt source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kI2C_GlobalInterruptEnable</li><li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li><li>• kI2C_SdaTimeoutInterruptEnable</li></ul>

### 17.2.7.13 void I2C\_DisableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

Parameters

<i>base</i>	I2C base pointer
<i>mask</i>	interrupt source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kI2C_GlobalInterruptEnable</li><li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li><li>• kI2C_SdaTimeoutInterruptEnable</li></ul>

### 17.2.7.14 void I2C\_MasterSetBaudRate ( I2C\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

Parameters

<i>base</i>	I2C base pointer
<i>baudRate_Bps</i>	the baud rate value in bps
<i>srcClock_Hz</i>	Source clock

### 17.2.7.15 status\_t I2C\_MasterStart ( *I2C\_Type \* base*, *uint8\_t address*, *i2c\_direction\_t direction* )

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

## I2C Driver

Parameters

<i>base</i>	I2C peripheral base pointer
<i>address</i>	7-bit slave device address.
<i>direction</i>	Master transfer directions(transmit/receive).

Return values

<i>kStatus_Success</i>	Successfully send the start signal.
<i>kStatus_I2C_Busy</i>	Current bus is busy.

### 17.2.7.16 status\_t I2C\_MasterStop ( I2C\_Type \* *base* )

Return values

<i>kStatus_Success</i>	Successfully send the stop signal.
<i>kStatus_I2C_Timeout</i>	Send stop signal failed, timeout.

### 17.2.7.17 status\_t I2C\_MasterRepeatedStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )

Parameters

<i>base</i>	I2C peripheral base pointer
<i>address</i>	7-bit slave device address.
<i>direction</i>	Master transfer directions(transmit/receive).

Return values

<i>kStatus_Success</i>	Successfully send the start signal.
<i>kStatus_I2C_Busy</i>	Current bus is busy but not occupied by current I2C master.

### 17.2.7.18 status\_t I2C\_MasterWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize*, uint32\_t *flags* )

## Parameters

<i>base</i>	The I2C peripheral base pointer.
<i>txBuff</i>	The pointer to the data to be transferred.
<i>txSize</i>	The length in bytes of the data to be transferred.
<i>flags</i>	Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

## Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_ArbitrationLost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

**17.2.7.19 status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize*, uint32\_t *flags* )**

## Note

The I2C\_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

## Parameters

<i>base</i>	I2C peripheral base pointer.
<i>rxBuff</i>	The pointer to the data to store the received data.
<i>rxSize</i>	The length in bytes of the data to be received.
<i>flags</i>	Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

## Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
------------------------	--

## I2C Driver

<code>kStatus_I2C_Timeout</code>	Send stop signal failed, timeout.
----------------------------------	-----------------------------------

**17.2.7.20 `status_t I2C_SlaveWriteBlocking ( I2C_Type * base, const uint8_t * txBuff, size_t txSize )`**

Parameters

<code>base</code>	The I2C peripheral base pointer.
<code>txBuff</code>	The pointer to the data to be transferred.
<code>txSize</code>	The length in bytes of the data to be transferred.

Return values

<code>kStatus_Success</code>	Successfully complete the data transmission.
<code>kStatus_I2C_ArbitrationLost</code>	Transfer error, arbitration lost.
<code>kStatus_I2C_Nak</code>	Transfer error, receive NAK during transfer.

**17.2.7.21 `void I2C_SlaveReadBlocking ( I2C_Type * base, uint8_t * rxBuff, size_t rxSize )`**

Parameters

<code>base</code>	I2C peripheral base pointer.
<code>rxBuff</code>	The pointer to the data to store the received data.
<code>rxSize</code>	The length in bytes of the data to be received.

**17.2.7.22 `status_t I2C_MasterTransferBlocking ( I2C_Type * base, i2c_master_transfer_t * xfer )`**

Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

<i>base</i>	I2C peripheral base address.
<i>xfer</i>	Pointer to the transfer structure.

Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.
<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.
<i>kStatus_I2C_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

#### 17.2.7.23 void I2C\_MasterTransferCreateHandle ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle, i2c\_master\_transfer\_callback\_t callback, void \* userData* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to <i>i2c_master_handle_t</i> structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

#### 17.2.7.24 status\_t I2C\_MasterTransferNonBlocking ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle, i2c\_master\_transfer\_t \* xfer* )

Note

Calling the API returns immediately after transfer initiates. The user needs to call *I2C\_MasterGetTransferCount* to poll the transfer status to check whether the transfer is finished. If the return status is not *kStatus\_I2C\_Busy*, the transfer is finished.

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to <i>i2c_master_handle_t</i> structure which stores the transfer state.
<i>xfer</i>	pointer to <i>i2c_master_transfer_t</i> structure.

## I2C Driver

Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.
<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.

**17.2.7.25 status\_t I2C\_MasterTransferGetCount ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle, size\_t \* count* )**

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to <i>i2c_master_handle_t</i> 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.

**17.2.7.26 void I2C\_MasterTransferAbort ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle* )**

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to <i>i2c_master_handle_t</i> structure which stores the transfer state

**17.2.7.27 void I2C\_MasterTransferHandleIRQ ( *I2C\_Type \* base, void \* i2cHandle* )**

Parameters

<i>base</i>	I2C base pointer.
<i>i2cHandle</i>	pointer to i2c_master_handle_t structure.

#### 17.2.7.28 void I2C\_SlaveTransferCreateHandle ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, i2c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

#### 17.2.7.29 status\_t I2C\_SlaveTransferNonBlocking ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, uint32\_t *eventMask* )

Call this API after calling the [I2C\\_SlaveInit\(\)](#) and [I2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to [I2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [k\\_I2C\\_SlaveTransmitEvent](#) and #[kLPI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>handle</i>	Pointer to # <a href="#">i2c_slave_handle_t</a> structure which stores the transfer state.
<i>eventMask</i>	Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events.

## I2C Driver

Return values

#kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

### 17.2.7.30 void I2C\_SlaveTransferAbort ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )

Note

This API can be called at any time to stop slave for handling the bus events.

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure which stores the transfer state.

### 17.2.7.31 status\_t I2C\_SlaveTransferGetCount ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

### 17.2.7.32 void I2C\_SlaveTransferHandleIRQ ( I2C\_Type \* *base*, void \* *i2cHandle* )

Parameters

<i>base</i>	I2C base pointer.
<i>i2cHandle</i>	pointer to i2c_slave_handle_t structure which stores the transfer state

## I2C FreeRTOS Driver

### 17.3 I2C FreeRTOS Driver

#### 17.3.1 Overview

#### I2C RTOS Operation

- status\_t [I2C\\_RTOS\\_Init](#) (i2c\_rtos\_handle\_t \*handle, I2C\_Type \*base, const i2c\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes I2C.*
- status\_t [I2C\\_RTOS\\_Deinit](#) (i2c\_rtos\_handle\_t \*handle)  
*Deinitializes the I2C.*
- status\_t [I2C\\_RTOS\\_Transfer](#) (i2c\_rtos\_handle\_t \*handle, i2c\_master\_transfer\_t \*transfer)  
*Performs the I2C transfer.*

#### 17.3.2 Function Documentation

##### 17.3.2.1 status\_t I2C\_RTOS\_Init ( *i2c\_rtos\_handle\_t \* handle, I2C\_Type \* base, const i2c\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz* )

This function initializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the I2C instance to initialize.
<i>masterConfig</i>	The configuration structure to set-up I2C in master mode.
<i>srcClock_Hz</i>	The frequency of an input clock of the I2C module.

Returns

status of the operation.

##### 17.3.2.2 status\_t I2C\_RTOS\_Deinit ( *i2c\_rtos\_handle\_t \* handle* )

This function deinitializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle.
---------------	----------------------

### 17.3.2.3 **status\_t I2C\_RTOS\_Transfer( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )**

This function performs the I2C transfer according to the data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS I2C handle.
<i>transfer</i>	A structure specifying the transfer parameters.

Returns

status of the operation.



# Chapter 18

## PWM: Pulse Width Modulation Driver

### 18.1 Overview

#### Modules

- [PWM Driver](#)

*Main function \*/ int main(void) { pwm\_config\_t pwmConfig;*

## PWM Driver

### 18.2 PWM Driver

Main function \*/ int main(void) { pwm\_config\_t pwmConfig;

#### 18.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Pulse Width Modulation (PWM) module of MCUXpresso SDK devices.

The function [PWM\\_Init\(\)](#) initializes the PWM with a specified configurations. The function [PWM\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the PWM for the requested register update mode for registers with buffers.

The function [PWM\\_Deinit\(\)](#) disables the PWM counter and turns off the module clock.

#### 18.2.2 Typical use case

##### 18.2.2.1 PWM output

Output PWM signal on PWM3 module with different dutycycles. Periodically update the PWM signal duty cycle.

```
void DEMO_PWM_IRQHandler(void)
{
    /* Gets interrupt kPWM_FIFOEmptyFlag */
    if(PWM_GetStatusFlags(DEMO_PWM_BASEADDR) & kPWM_FIFOEmptyFlag)
    {
        if(pwmDutyUp)
        {
            /* Increase duty cycle until it reach limited value. */
            if(++pwmDutycycle > PWM_PERIOD_VALUE)
            {
                pwmDutycycle = PWM_PERIOD_VALUE;
                pwmDutyUp = false;
            }
        }
        else
        {
            /* Decrease duty cycle until it reach limited value. */
            if(--pwmDutycycle == 0U)
            {
                pwmDutyUp = true;
            }
        }
        /* Write duty cycle to PWM sample register. */
        PWM_SetSampleValue(DEMO_PWM_BASEADDR, pwmDutycycle);
        /* Clear kPWM_FIFOEmptyFlag */
        PWM_clearStatusFlags(DEMO_PWM_BASEADDR, kPWM_FIFOEmptyFlag);
    }
}

BOARD_InitHardware();
SystemInstallIrqHandler(DEMO_PWM_IRQn, (system_irq_handler_t)DEMO_PWM_IRQHandler, NULL);

PRINTF("\r\nPWM driver example.\r\n");
```

```

PWM_GetDefaultConfig(&pwmConfig);

/* Initialize PWM module */
PWM_Init(DEMO_PWM_BASEADDR, &pwmConfig);

/* Enable FIFO empty interrupt */
PWM_EnableInterrupts(DEMO_PWM_BASEADDR,
    kPWM_FIFOEmptyInterruptEnable);

/* Three initial samples be written to the PWM Sample Register */
for(pwmDutycycle = 0U; pwmDutycycle < 3; pwmDutycycle++)
{
    PWM_SetSampleValue(DEMO_PWM_BASEADDR, pwmDutycycle);
}

/* Check and Clear interrupt status flags */
if(PWM_GetStatusFlags(DEMO_PWM_BASEADDR))
{
    PWM_clearStatusFlags(DEMO_PWM_BASEADDR, kPWM_FIFOEmptyFlag |
    kPWM_RolloverFlag | kPWM_CompareFlag |
    kPWM_FIFOWriteErrorFlag);
}

/* Write the period to the PWM Period Register */
PWM_SetPeriodValue(DEMO_PWM_BASEADDR, PWM_PERIOD_VALUE);

/* Enable PWM interrupt request */
EnableIRQ(DEMO_PWM IRQn);

PWM_StartTimer(DEMO_PWM_BASEADDR);

while (1)
{
}
}
}

```

## Enumerations

- enum `pwm_clock_source_t` {
   
kPWM\_PeripheralClock = 0U,
   
kPWM\_HighFrequencyClock,
   
kPWM\_LowFrequencyClock }
   
*PWM clock source select.*
- enum `pwm_fifo_water_mark_t` {
   
kPWM\_FIFOWaterMark\_1 = 0U,
   
kPWM\_FIFOWaterMark\_2,
   
kPWM\_FIFOWaterMark\_3,
   
kPWM\_FIFOWaterMark\_4 }
   
*PWM FIFO water mark select.*
- enum `pwm_byte_data_swap_t` {
   
kPWM\_ByteNoSwap = 0U,
   
kPWM\_ByteSwap }
   
*PWM byte data swap select.*
- enum `pwm_half_word_data_swap_t` {
   
kPWM\_HalfWordNoSwap = 0U,
   
kPWM\_HalfWordSwap }
   
*PWM half-word data swap select.*

## PWM Driver

- enum `pwm_output_configuration_t` {  
    `kPWM_SetAtRolloverAndClearAtcomparison` = 0U,  
    `kPWM_ClearAtRolloverAndSetAtcomparison`,  
    `kPWM_NoConfigure` }  
    *PWM Output Configuration.*
- enum `pwm_sample_repeat_t` {  
    `kPWM_EachSampleOnce` = 0u,  
    `kPWM_EachSamplentwice`,  
    `kPWM_EachSampleFourTimes`,  
    `kPWM_EachSampleEightTimes` }  
    *PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.*
- enum `pwm_interrupt_enable_t` {  
    `kPWM_FIFOEmptyInterruptEnable` = (1U << 0),  
    `kPWM_RolloverInterruptEnable` = (1U << 1),  
    `kPWM_CompareInterruptEnable` = (1U << 2) }  
    *List of PWM interrupt options.*
- enum `pwm_status_flags_t` {  
    `kPWM_FIFOEmptyFlag` = (1U << 3),  
    `kPWM_RolloverFlag` = (1U << 4),  
    `kPWM_CompareFlag` = (1U << 5),  
    `kPWM_FIFOWriteErrorFlag` = (1U << 6) }  
    *List of PWM status flags.*
- enum `pwm_fifo_available_t` {  
    `kPWM_NoDataInFIFOFlag` = 0U,  
    `kPWM_OneWordInFIFOFlag`,  
    `kPWM_TwoWordsInFIFOFlag`,  
    `kPWM_ThreeWordsInFIFOFlag`,  
    `kPWM_FourWordsInFIFOFlag` }  
    *List of PWM FIFO available.*

## Functions

- static void `PWM_SoftwareReset` (PWM\_Type \*base)  
    *Sofrware reset.*
- static void `PWM_SetPeriodValue` (PWM\_Type \*base, uint32\_t value)  
    *Sets the PWM period value.*
- static uint32\_t `PWM_GetPeriodValue` (PWM\_Type \*base)  
    *Gets the PWM period value.*
- static uint32\_t `PWM_GetCounterValue` (PWM\_Type \*base)  
    *Gets the PWM counter value.*

## 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, const pwm\_config\_t \*config)  
*Ungates the PWM clock and configures the peripheral for basic operation.*
- void **PWM\_Deinit** (PWM\_Type \*base)  
*Gate the PWM submodule clock.*
- void **PWM\_GetDefaultConfig** (pwm\_config\_t \*config)  
*Fill in the PWM config struct with the default settings.*

## PWM start and stop.

- static void **PWM\_StartTimer** (PWM\_Type \*base)  
*Starts the PWM counter when the PWM is enabled.*
- static void **PWM\_StopTimer** (PWM\_Type \*base)  
*Stops the PWM counter when the pwm is disabled.*

## Interrupt Interface

- static void **PWM\_EnableInterrupts** (PWM\_Type \*base, uint32\_t mask)  
*Enables the selected PWM interrupts.*
- static void **PWM\_DisableInterrupts** (PWM\_Type \*base, uint32\_t mask)  
*Disables the selected PWM interrupts.*
- static uint32\_t **PWM\_GetEnabledInterrupts** (PWM\_Type \*base)  
*Gets the enabled PWM interrupts.*

## Status Interface

- static uint32\_t **PWM\_GetStatusFlags** (PWM\_Type \*base)  
*Gets the PWM status flags.*
- static void **PWM\_clearStatusFlags** (PWM\_Type \*base, uint32\_t mask)  
*Clears the PWM status flags.*
- static uint32\_t **PWM\_GetFIFOAvailable** (PWM\_Type \*base)  
*Gets the PWM FIFO available.*

## Sample Interface

- static void **PWM\_SetSampleValue** (PWM\_Type \*base, uint32\_t value)  
*Sets the PWM sample value.*
- static uint32\_t **PWM\_GetSampleValue** (PWM\_Type \*base)  
*Gets the PWM sample value.*

### 18.2.3 Enumeration Type Documentation

#### 18.2.3.1 enum pwm\_clock\_source\_t

Enumerator

**kPWM\_PeripheralClock** The Peripheral clock is used as the clock.

**kPWM\_HighFrequencyClock** High-frequency reference clock is used as the clock.

**kPWM\_LowFrequencyClock** Low-frequency reference clock(32KHz) is used as the clock.

#### 18.2.3.2 enum pwm\_fifo\_water\_mark\_t

Sets the data level at which the FIFO empty flag will be set

Enumerator

**kPWM\_FIFOWaterMark\_1** FIFO empty flag is set when there are more than or equal to 1 empty slots.

**kPWM\_FIFOWaterMark\_2** FIFO empty flag is set when there are more than or equal to 2 empty slots.

**kPWM\_FIFOWaterMark\_3** FIFO empty flag is set when there are more than or equal to 3 empty slots.

**kPWM\_FIFOWaterMark\_4** FIFO empty flag is set when there are more than or equal to 4 empty slots.

#### 18.2.3.3 enum pwm\_byte\_data\_swap\_t

It determines the byte ordering of the 16-bit data when it goes into the FIFO from the sample register.

Enumerator

**kPWM\_ByteNoSwap** byte ordering remains the same

**kPWM\_ByteSwap** byte ordering is reversed

#### 18.2.3.4 enum pwm\_half\_word\_data\_swap\_t

Enumerator

**kPWM\_HalfWordNoSwap** Half word swapping does not take place.

**kPWM\_HalfWordSwap** Half word from write data bus are swapped.

### 18.2.3.5 enum pwm\_output\_configuration\_t

Enumerator

- kPWM\_SetAtRolloverAndClearAtcomparison*** Output pin is set at rollover and cleared at comparison.
- kPWM\_ClearAtRolloverAndSetAtcomparison*** Output pin is cleared at rollover and set at comparison.
- kPWM\_NoConfigure*** PWM output is disconnected.

### 18.2.3.6 enum pwm\_sample\_repeat\_t

Enumerator

- kPWM\_EachSampleOnce*** Use each sample once.
- kPWM\_EachSampleTwice*** Use each sample twice.
- kPWM\_EachSampleFourTimes*** Use each sample four times.
- kPWM\_EachSampleEightTimes*** Use each sample eight times.

### 18.2.3.7 enum pwm\_interrupt\_enable\_t

Enumerator

- kPWM\_FIFOEmptyInterruptEnable*** This bit controls the generation of the FIFO Empty interrupt.
- kPWM\_RolloverInterruptEnable*** This bit controls the generation of the Rollover interrupt.
- kPWM\_CompareInterruptEnable*** This bit controls the generation of the Compare interrupt.

### 18.2.3.8 enum pwm\_status\_flags\_t

Enumerator

- kPWM\_FIFOEmptyFlag*** This bit indicates the FIFO data level in comparison to the water level set by FWM field in the control register.
- kPWM\_RolloverFlag*** This bit shows that a roll-over event has occurred.
- kPWM\_CompareFlag*** This bit shows that a compare event has occurred.
- kPWM\_FIFOWriteErrorFlag*** This bit shows that an attempt has been made to write FIFO when it is full.

### 18.2.3.9 enum pwm\_fifo\_available\_t

Enumerator

- kPWM\_NoDataInFIFOFlag*** No data available.

## PWM Driver

- kPWM\_OneWordInFIFOFlag* 1 word of data in FIFO
- kPWM\_TwoWordsInFIFOFlag* 2 word of data in FIFO
- kPWM\_ThreeWordsInFIFOFlag* 3 word of data in FIFO
- kPWM\_FourWordsInFIFOFlag* 4 word of data in FIFO

### 18.2.4 Function Documentation

#### 18.2.4.1 status\_t PWM\_Init ( **PWM\_Type** \* *base*, **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>config</i>	Pointer to user's PWM config structure.

Returns

kStatus\_Success means success; else failed.

#### 18.2.4.2 void PWM\_Deinit ( **PWM\_Type** \* *base* )

Parameters

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

#### 18.2.4.3 void PWM\_GetDefaultConfig ( **pwm\_config\_t** \* *config* )

The default values are:

```
* config->enableStopMode = false;
* config->enableDozeMode = false;
* config->enableWaitMode = false;
* config->enableDozeMode = false;
* config->clockSource = kPWM_LowFrequencyClock;
* config->prescale = 0U;
* config->outputConfig = kPWM_SetAtRolloverAndClearAtComparison;
* config->fifoWater = kPWM_FIFOWaterMark_2;
* config->sampleRepeat = kPWM_EachSampleOnce;
* config->byteSwap = kPWM_ByteNoSwap;
* config->halfWordSwap = kPWM_HalfWordNoSwap;
*
```

Parameters

<i>config</i>	Pointer to user's PWM config structure.
---------------	---

#### 18.2.4.4 static void PWM\_StartTimer ( PWM\_Type \* *base* ) [inline], [static]

When the PWM is enabled, it begins a new period, the output pin is set to start a new period while the prescaler and counter are released and counting begins.

Parameters

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

#### 18.2.4.5 static void PWM\_StopTimer ( PWM\_Type \* *base* ) [inline], [static]

Parameters

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

#### 18.2.4.6 static void PWM\_SoftwareReset ( PWM\_Type \* *base* ) [inline], [static]

PWM is reset when this bit is set to 1. It is a self clearing bit. Setting this bit resets all the registers to their reset values except for the STOPEN, DOZEN, WAITEN, and DBGEN bits in this control register.

Parameters

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

#### 18.2.4.7 static void PWM\_EnableInterrupts ( PWM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	PWM peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a>

**18.2.4.8 static void PWM\_DisableInterrupts ( *PWM\_Type* \* *base*, *uint32\_t mask* )  
[inline], [static]**

Parameters

<i>base</i>	PWM peripheral base address
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a>

#### 18.2.4.9 static uint32\_t PWM\_GetEnabledInterrupts ( PWM\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [pwm\\_interrupt\\_enable\\_t](#)

#### 18.2.4.10 static uint32\_t PWM\_GetStatusFlags ( PWM\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

#### 18.2.4.11 static void PWM\_clearStatusFlags ( PWM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

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

## PWM Driver

<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>
-------------	--

**18.2.4.12 static uint32\_t PWM\_GetFIFOAvailable ( PWM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [pwm\\_fifo\\_available\\_t](#)

**18.2.4.13 static void PWM\_SetSampleValue ( PWM\_Type \* *base*, uint32\_t *value* ) [inline], [static]**

Parameters

<i>base</i>	PWM peripheral base address
<i>mask</i>	The sample value. This is the input to the 4x16 FIFO. The value in this register denotes the value of the sample being currently used.

**18.2.4.14 static uint32\_t PWM\_GetSampleValue ( PWM\_Type \* *base* ) [static]**

Parameters

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

Returns

The sample value. It can be read only when the PWM is enable.

**18.2.4.15 static void PWM\_SetPeriodValue ( PWM\_Type \* *base*, uint32\_t *value* ) [inline], [static]**

Parameters

<i>base</i>	PWM peripheral base address
<i>mask</i>	The period value. The PWM period register (PWM_PWMMPR) determines the period of the PWM output signal. Writing 0xFFFF to this register will achieve the same result as writing 0xFFFE. PWMO (Hz) = PCLK(Hz) / (period +2)

**18.2.4.16 static uint32\_t PWM\_GetPeriodValue ( PWM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The period value. The PWM period register (PWM\_PWMMPR) determines the period of the PWM output signal.

**18.2.4.17 static uint32\_t PWM\_GetCounterValue ( PWM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The counter value. The current count value.



# Chapter 19

## UART: Universal Asynchronous Receiver/Transmitter Driver

### 19.1 Overview

#### Modules

- UART Driver
- UART FreeRTOS Driver

### 19.2 UART Driver

#### 19.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) module of MCUXpresso SDK devices.

The UART driver includes functional APIs and transactional APIs.

Functional APIs are used for UART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the UART peripheral and how to organize functional APIs to meet the application requirements. All functional APIs use the peripheral base address as the first parameter. UART functional operation groups provide the functional API set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `uart_handle_t` as the second parameter. Initialize the handle by calling the [UART\\_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer, which means that the functions [UART\\_TransferSendNonBlocking\(\)](#) and [UART\\_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_UART_TxIdle` and `kStatus_UART_RxIdle`.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the [UART\\_TransferCreateHandle\(\)](#). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The [UART\\_TransferReceiveNonBlocking\(\)](#) function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `kStatus_UART_RxIdle`.

If the receive ring buffer is full, the upper layer is informed through a callback with the `kStatus_UART_RxRingBufferOverrun`. In the callback function, the upper layer reads data out from the ring buffer. If not, existing data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code.

```
UART_TransferCreateHandle(UART0, &handle, UART_UserCallback, NULL);
```

In this example, the buffer size is 32, but only 31 bytes are used for saving data.

#### 19.2.2 Typical use case

##### 19.2.2.1 UART Send/receive using a polling method

```
UART_GetDefaultConfig(&config);
```

```

config.baudRate_Bps = DEMO_UART_BAUDRATE;
config.enableTx = true;
config.enableRx = true;

UART_Init(DEMO_UART, &config, DEMO_UART_FREQ);
UART_WriteBlocking(DEMO_UART, txbuff, sizeof(txbuff) - 1);

while (1)
{
    UART_ReadBlocking(DEMO_UART, &ch, 1);
    UART_WriteBlocking(DEMO_UART, &ch, 1);
}

```

### 19.2.2.2 UART Send/receive using an interrupt method

```

void DEMO_UART_IRQHandler(void)
{
    uint8_t data;

    /* If new data arrived. */
    if ((UART_GetStatusFlag(DEMO_UART, kUART_RxDataReadyFlag)) || (
        UART_GetStatusFlag(DEMO_UART, kUART_RxOverrunFlag)))
    {
        data = UART_ReadByte(DEMO_UART);

        /* If ring buffer is not full, add data to ring buffer. */
        if (((rxIndex + 1) % DEMO_RING_BUFFER_SIZE) != txIndex)
        {
            demoRingBuffer[rxIndex] = data;
            rxIndex++;
            rxIndex %= DEMO_RING_BUFFER_SIZE;
        }
    }
}

int main(void)
{
    uart_config_t config;

    BOARD_InitHardware();
    SystemInstallIrqHandler(DEMO_IRQn, (system_irq_handler_t)DEMO_UART_IRQHandler, NULL);

    /*
     * config.baudRate_Bps = 115200U;
     * config.parityMode = kUART_ParityDisabled;
     * config.dataBitsCount = kUART_EightDataBits;
     * config.stopBitCount = kUART_OneStopBit;
     * config.txFifoWatermark = 2;
     * config.rxFifoWatermark = 1;
     * config.enableTx = false;
     * config.enableRx = false;
     */
    UART_GetDefaultConfig(&config);
    config.baudRate_Bps = DEMO_UART_BAUDRATE;
    config.enableTx = true;
    config.enableRx = true;

    UART_Init(DEMO_UART, &config, DEMO_UART_CLKSRC);

    /* Send g_tipString out. */
    UART_WriteBlocking(DEMO_UART, g_tipString, sizeof(g_tipString) / sizeof(g_tipString[0])
        ) - 1);

    /* Enable RX interrupt. */
    UART_EnableInterrupts(DEMO_UART, kUART_RxDataReadyEnable | kUART_RxOverrunEnable);

```

## UART Driver

```
EnableIRQ(DEMO_IRQn);

while (1)
{
    /* Send data only when UART TX register is empty and ring buffer has data to send out. */
    while ((UART_GetStatusFlag(DEMO_UART,
        kUART_TxReadyFlag)) && (rxIndex != txIndex))
    {
        UART_WriteByte(DEMO_UART, demoRingBuffer[txIndex]);
        txIndex++;
        txIndex %= DEMO_RING_BUFFER_SIZE;
    }
}
```

### 19.2.2.3 UART Receive using the ringbuffer feature

```
#define RX_RING_BUFFER_SIZE 20U
#define ECHO_BUFFER_SIZE 8U

/*****
 * Prototypes
 ****/
/* UART user callback */
void UART_UserCallback(UART_Type *base, uart_handle_t *handle, status_t status, void *userData);

/*****
 * Variables
 ****/
uart_handle_t g_uartHandle;
uint8_t g_tipString[] = "UART RX ring buffer example\r\nSend back received data\r\nEcho every 8 bytes\r\n";
uint8_t g_rxRingBuffer[RX_RING_BUFFER_SIZE] = {0}; /* RX ring buffer. */

uint8_t g_rxBuffer[ECHO_BUFFER_SIZE] = {0}; /* Buffer for receive data to echo. */
uint8_t g_txBuffer[ECHO_BUFFER_SIZE] = {0}; /* Buffer for send data to echo. */
volatile bool rxBufferEmpty = true;
volatile bool txBufferFull = false;
volatile bool txOnGoing = false;
volatile bool rxOnGoing = false;

/*****
 * Code
 ****/
/* UART user callback */
void UART_UserCallback(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_UART_TxIdle == status)
    {
        txBufferFull = false;
        txOnGoing = false;
    }
}

int main(void)
{
    uart_config_t config;
    uart_transfer_t xfer;
    uart_transfer_t sendXfer;
    uart_transfer_t receiveXfer;
    size_t receivedBytes;
    uint32_t i;
```

```

BOARD_InitHardware();

/*
 * config.baudRate_Bps = 115200U;
 * config.parityMode = kUART_ParityDisabled;
 * config.dataBitsCount = kUART_EightDataBits;
 * config.stopBitCount = kUART_OneStopBit;
 * config.txFifoWatermark = 2;
 * config.rxFifoWatermark = 16;
 * config.enableTx = false;
 * config.enableRx = false;
 */
UART_GetDefaultConfig(&config);
config.baudRate_Bps = DEMO_UART_BAUDRATE;
config.txFifoWatermark = 2;
config.rxFifoWatermark = 16;
config.enableTx = true;
config.enableRx = true;

UART_Init(DEMO_UART, &config, DEMO_UART_CLKSRC);
UART_TransferCreateHandle(DEMO_UART, &g_uartHandle, UART_UserCallback, NULL);
UART_TransferStartRingBuffer(DEMO_UART, &g_uartHandle, g_rxRingBuffer,
    RX_RING_BUFFER_SIZE);

/* Send g_tipString out. */
xfer.data = g_tipString;
xfer.dataSize = sizeof(g_tipString) - 1;
txOnGoing = true;
UART_TransferSendNonBlocking(DEMO_UART, &g_uartHandle, &xfer);

/* Wait send finished */
while (txOnGoing)
{
}

/* Start to echo. */
sendXfer.data = g_txBuffer;
sendXfer.dataSize = ECHO_BUFFER_SIZE;
receiveXfer.data = g_rxBuffer;
receiveXfer.dataSize = ECHO_BUFFER_SIZE;

while (1)
{
    /* If g_txBuffer is empty and g_rxBuffer is full, copy g_rxBuffer to g_txBuffer. */
    if ((!rxBufferEmpty) && (!txBufferFull))
    {
        memcpy(g_txBuffer, g_rxBuffer, ECHO_BUFFER_SIZE);
        rxBufferEmpty = true;
        txBufferFull = true;
    }

    /* If the data in ring buffer reach ECHO_BUFFER_SIZE, then start to read data from ring buffer. */
    if (ECHO_BUFFER_SIZE <= UART_TransferGetRxRingBufferLength(&
g_uartHandle))
    {
        UART_TransferReceiveNonBlocking(DEMO_UART, &g_uartHandle, &
receiveXfer, &receivedBytes);
        rxBufferEmpty = false;
    }

    /* If TX is idle and g_txBuffer is full, start to send data. */
    if (!txOnGoing) && txBufferFull)
    {
        txOnGoing = true;
        UART_TransferSendNonBlocking(DEMO_UART, &g_uartHandle, &sendXfer);
    }

    /* Delay some time, simulate the app is processing other things, input data save to ring buffer. */
}

```

## UART Driver

```
i = 0x10U;
while (i--)
{
    __NOP();
}
}
```

### 19.2.2.4 UART automatic baud rate detect feature

```
int main(void)
{
    uint8_t ch;

    uart_config_t config;

    BOARD_InitHardware();

    /*
     * config.baudRate_Bps = 115200U;
     * config.parityMode = kUART_ParityDisabled;
     * config.stopBitCount = kUART_OneStopBit;
     * config.txFifoWatermark = 2;
     * config.rxFifoWatermark = 1;
     * config.enableTx = false;
     * config.enableRx = false;
     */
    UART_GetDefaultConfig(&config);
    config.baudRate_Bps = NULL;

    config.enableTx = true;
    config.enableRx = true;

    UART_Init(DEMO_UART, &config, DEMO_UART_FREQ);
    UART_EnableAutoBaudRate(DEMO_UART, true);

    while (!UART_IsAutoBaudRateComplete(DEMO_UART))
    {
        UART_WriteBlocking(DEMO_UART, infobuff, sizeof(infobuff) - 1);
        /* Read the detect character from receiver register */
        UART_ReadBlocking(DEMO_UART, &ch, 1);
        UART_WriteBlocking(DEMO_UART, &ch, 1);

        UART_WriteBlocking(DEMO_UART, txbuff, sizeof(txbuff) - 1);

        while (1)
        {
            UART_ReadBlocking(DEMO_UART, &ch, 1);
            UART_WriteBlocking(DEMO_UART, &ch, 1);
        }
    }
}
```

## Data Structures

- struct **uart\_config\_t**  
*UART configuration structure.* [More...](#)
- struct **uart\_transfer\_t**  
*UART transfer structure.* [More...](#)
- struct **uart\_handle\_t**  
*UART handle structure.* [More...](#)

## Typedefs

- `typedef void(* uart_transfer_callback_t )(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)`  
*UART transfer callback function.*

## Enumerations

- `enum _uart_status {`  
 `kStatus_UART_TxBusy = MAKE_STATUS(kStatusGroup_IUART, 0),`  
 `kStatus_UART_RxBusy = MAKE_STATUS(kStatusGroup_IUART, 1),`  
 `kStatus_UART_TxIdle = MAKE_STATUS(kStatusGroup_IUART, 2),`  
 `kStatus_UART_RxIdle = MAKE_STATUS(kStatusGroup_IUART, 3),`  
 `kStatus_UART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_IUART, 4),`  
 `kStatus_UART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_IUART, 5),`  
 `kStatus_UART_FlagCannotClearManually,`  
 `kStatus_UART_Error = MAKE_STATUS(kStatusGroup_IUART, 7),`  
 `kStatus_UART_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_IUART, 8),`  
 `kStatus_UART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_IUART, 9),`  
 `kStatus_UART_NoiseError = MAKE_STATUS(kStatusGroup_IUART, 10),`  
 `kStatus_UART_FramingError = MAKE_STATUS(kStatusGroup_IUART, 11),`  
 `kStatus_UART_ParityError = MAKE_STATUS(kStatusGroup_IUART, 12),`  
 `kStatus_UART_BaudrateNotSupport,`  
 `kStatus_UART_BreakDetect = MAKE_STATUS(kStatusGroup_IUART, 14) }`  
*Error codes for the UART driver.*
- `enum uart_data_bits_t {`  
 `kUART_SevenDataBits = 0x0U,`  
 `kUART_EightDataBits = 0x1U }`  
*UART data bits count.*
- `enum uart_parity_mode_t {`  
 `kUART_ParityDisabled = 0x0U,`  
 `kUART_ParityEven = 0x2U,`  
 `kUART_ParityOdd = 0x3U }`  
*UART parity mode.*
- `enum uart_stop_bit_count_t {`  
 `kUART_OneStopBit = 0x0U,`  
 `kUART_TwoStopBit = 0x1U }`  
*UART stop bit count.*
- `enum _uart_interrupt_enable`  
*This structure contains the settings for all of the UART interrupt configurations.*
- `enum _uart_flags {`

## UART Driver

```
kUART_RxCharReadyFlag = 0x0000000FU,  
kUART_RxErrorFlag = 0x0000000EU,  
kUART_RxOverrunErrorFlag = 0x0000000DU,  
kUART_RxFrameErrorFlag = 0x0000000CU,  
kUART_RxBreakDetectFlag = 0x0000000BU,  
kUART_RxParityErrorFlag = 0x0000000AU,  
kUART_ParityErrorFlag = 0x0094000FU,  
kUART_RtsStatusFlag = 0x0094000EU,  
kUART_TxReadyFlag = 0x0094000DU,  
kUART_RtsDeltaFlag = 0x0094000CU,  
kUART_EscapeFlag = 0x0094000BU,  
kUART_FrameErrorFlag = 0x0094000AU,  
kUART_RxReadyFlag = 0x00940009U,  
kUART_AgingTimerFlag = 0x00940008U,  
kUART_DtrDeltaFlag = 0x00940007U,  
kUART_RxDsFlag = 0x00940006U,  
kUART_tAWakeFlag = 0x00940005U,  
kUART_AwakeFlag = 0x00940004U,  
kUART_Rs485SlaveAddrMatchFlag = 0x00940003U,  
kUART_AutoBaudFlag = 0x0098000FU,  
kUART_TxEmptyFlag = 0x0098000EU,  
kUART_DtrFlag = 0x0098000DU,  
kUART_IdleFlag = 0x0098000CU,  
kUART_AutoBaudCntStopFlag = 0x0098000BU,  
kUART_RiDeltaFlag = 0x0098000AU,  
kUART_RiFlag = 0x00980009U,  
kUART_IrFlag = 0x00980008U,  
kUART_WakeFlag = 0x00980007U,  
kUART_DcdDeltaFlag = 0x00980006U,  
kUART_DcdFlag = 0x00980005U,  
kUART_RtsFlag = 0x00980004U,  
kUART_TxCompleteFlag = 0x00980003U,  
kUART_BreakDetectFlag = 0x00980002U,  
kUART_RxOverrunFlag = 0x00980001U,  
kUART_RxDataReadyFlag = 0x00980000U }
```

*UART status flags.*

## Variables

- `uint32_t uart_config_t::baudRate_Bps`  
*UART baud rate.*
- `uart_parity_mode_t uart_config_t::parityMode`  
*Parity error check mode of this module.*
- `uart_data_bits_t uart_config_t::dataBitsCount`  
*Data bits count, eight (default), seven.*

- **uart\_stop\_bit\_count\_t uart\_config\_t::stopBitCount**  
*Number of stop bits in one frame.*
- **uint8\_t uart\_config\_t::txFifoWatermark**  
*TX FIFO watermark.*
- **uint8\_t uart\_config\_t::rxFifoWatermark**  
*RX FIFO watermark.*
- **bool uart\_config\_t::enableAutoBaudRate**  
*Enable automatic baud rate detection.*
- **bool uart\_config\_t::enableTx**  
*Enable TX.*
- **bool uart\_config\_t::enableRx**  
*Enable RX.*
- **uint8\_t \* uart\_transfer\_t::data**  
*The buffer of data to be transfer.*
- **size\_t uart\_transfer\_t::dataSize**  
*The byte count to be transfer.*
- **uint8\_t \*volatile uart\_handle\_t::txData**  
*Address of remaining data to send.*
- **volatile size\_t uart\_handle\_t::txDataSize**  
*Size of the remaining data to send.*
- **size\_t uart\_handle\_t::txDataSizeAll**  
*Size of the data to send out.*
- **uint8\_t \*volatile uart\_handle\_t::rxData**  
*Address of remaining data to receive.*
- **volatile size\_t uart\_handle\_t::rxDataSize**  
*Size of the remaining data to receive.*
- **size\_t uart\_handle\_t::rxDataSizeAll**  
*Size of the data to receive.*
- **uint8\_t \* uart\_handle\_t::rxRingBuffer**  
*Start address of the receiver ring buffer.*
- **size\_t uart\_handle\_t::rxRingBufferSize**  
*Size of the ring buffer.*
- **volatile uint16\_t uart\_handle\_t::rxRingBufferHead**  
*Index for the driver to store received data into ring buffer.*
- **volatile uint16\_t uart\_handle\_t::rxRingBufferTail**  
*Index for the user to get data from the ring buffer.*
- **uart\_transfer\_callback\_t uart\_handle\_t::callback**  
*Callback function.*
- **void \* uart\_handle\_t::userData**  
*UART callback function parameter.*
- **volatile uint8\_t uart\_handle\_t::txState**  
*TX transfer state.*
- **volatile uint8\_t uart\_handle\_t::rxState**  
*RX transfer state.*

## Driver version

- #define **FSL\_UART\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 0))  
*UART driver version 2.0.0.*

## UART Driver

### Software Reset

- static void **UART\_SoftwareReset** (UART\_Type \*base)  
*Resets the UART using software.*

### Initialization and deinitialization

- status\_t **UART\_Init** (UART\_Type \*base, const **uart\_config\_t** \*config, uint32\_t srcClock\_Hz)  
*Initializes an UART instance with the user configuration structure and the peripheral clock.*
- void **UART\_Deinit** (UART\_Type \*base)  
*Deinitializes a UART instance.*
- void **UART\_GetDefaultConfig** (**uart\_config\_t** \*config)  
*l*
- status\_t **UART\_SetBaudRate** (UART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the UART instance baud rate.*
- static void **UART\_Enable** (UART\_Type \*base)  
*This function is used to Enable the UART Module.*
- static void **UART\_Disable** (UART\_Type \*base)  
*This function is used to Disable the UART Module.*

### Status

- bool **UART\_GetStatusFlag** (UART\_Type \*base, uint32\_t flag)  
*This function is used to get the current status of specific UART status flag(including interrupt flag).*
- void **UART\_ClearStatusFlag** (UART\_Type \*base, uint32\_t flag)  
*This function is used to clear the current status of specific UART status flag.*

### Interrupts

- void **UART\_EnableInterrupts** (UART\_Type \*base, uint32\_t mask)  
*Enables UART interrupts according to the provided mask.*
- void **UART\_DisableInterrupts** (UART\_Type \*base, uint32\_t mask)  
*Disables the UART interrupts according to the provided mask.*
- uint32\_t **UART\_GetEnabledInterrupts** (UART\_Type \*base)  
*Gets enabled UART interrupts.*

### Bus Operations

- static void **UART\_EnableTx** (UART\_Type \*base, bool enable)  
*Enables or disables the UART transmitter.*
- static void **UART\_EnableRx** (UART\_Type \*base, bool enable)  
*Enables or disables the UART receiver.*
- static void **UART\_WriteByte** (UART\_Type \*base, uint8\_t data)  
*Writes to the transmitter register.*
- static uint8\_t **UART\_ReadByte** (UART\_Type \*base)

- *Reads the receiver register.*  
void [UART\\_WriteBlocking](#) (UART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*
- status\_t [UART\\_ReadBlocking](#) (UART\_Type \*base, uint8\_t \*data, size\_t length)  
*Read RX data register using a blocking method.*

## Transactional

- void [UART\\_TransferCreateHandle](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the UART handle.*
- void [UART\\_TransferStartRingBuffer](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void [UART\\_TransferStopRingBuffer](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- size\_t [UART\\_TransferGetRxRingBufferLength](#) (uart\_handle\_t \*handle)  
*Get the length of received data in RX ring buffer.*
- status\_t [UART\\_TransferSendNonBlocking](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void [UART\\_TransferAbortSend](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- status\_t [UART\\_TransferGetSendCount](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the UART TX register.*
- status\_t [UART\\_TransferReceiveNonBlocking](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using an interrupt method.*
- void [UART\\_TransferAbortReceive](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the interrupt-driven data receiving.*
- status\_t [UART\\_TransferGetReceiveCount](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes that have been received.*
- void [UART\\_TransferHandleIRQ](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*UART IRQ handle function.*

## DMA control functions.

- static void [UART\\_EnableTxDMA](#) (UART\_Type \*base, bool enable)  
*Enables or disables the UART transmitter DMA request.*
- static void [UART\\_EnableRxDMA](#) (UART\_Type \*base, bool enable)  
*Enables or disables the UART receiver DMA request.*

## UART Driver

### FIFO control functions.

- static void **UART\_SetTxFifoWatermark** (UART\_Type \*base, uint8\_t watermark)  
*This function is used to set the watermark of UART Tx FIFO.*
- static void **UART\_SetRxFifoWatermark** (UART\_Type \*base, uint8\_t watermark)  
*This function is used to set the watermark of UART Rx FIFO.*

### Auto baud rate detection.

- static void **UART\_EnableAutoBaudRate** (UART\_Type \*base, bool enable)  
*This function is used to set the enable condition of Automatic Baud Rate Detection feature.*
- static bool **UART\_IsAutoBaudRateComplete** (UART\_Type \*base)  
*This function is used to read if the automatic baud rate detection has finished.*

### 19.2.3 Data Structure Documentation

#### 19.2.3.1 struct uart\_config\_t

##### Data Fields

- uint32\_t **baudRate\_Bps**  
*UART baud rate.*
- **uart\_parity\_mode\_t parityMode**  
*Parity error check mode of this module.*
- **uart\_data\_bits\_t dataBitsCount**  
*Data bits count, eight (default), seven.*
- **uart\_stop\_bit\_count\_t stopBitCount**  
*Number of stop bits in one frame.*
- uint8\_t **txFifoWatermark**  
*TX FIFO watermark.*
- uint8\_t **rxFifoWatermark**  
*RX FIFO watermark.*
- bool **enableAutoBaudRate**  
*Enable automatic baud rate detection.*
- bool **enableTx**  
*Enable TX.*
- bool **enableRx**  
*Enable RX.*

#### 19.2.3.2 struct uart\_transfer\_t

##### Data Fields

- uint8\_t \* **data**  
*The buffer of data to be transfer.*
- size\_t **dataSize**  
*The byte count to be transfer.*

### 19.2.3.3 struct \_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.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `uint8_t *volatile rxData`  
*Address of remaining data to receive.*
- `volatile size_t rxDataSize`  
*Size of the remaining data to receive.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*
- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `uart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*UART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*

### 19.2.4 Macro Definition Documentation

19.2.4.1 `#define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

### 19.2.5 Typedef Documentation

19.2.5.1 `typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)`

### 19.2.6 Enumeration Type Documentation

#### 19.2.6.1 enum \_uart\_status

Enumerator

*kStatus\_UART\_TxBusy* Transmitter is busy.

*kStatus\_UART\_RxBusy* Receiver is busy.

*kStatus\_UART\_TxIdle* UART transmitter is idle.

*kStatus\_UART\_RxIdle* UART receiver is idle.

*kStatus\_UART\_TxWatermarkTooLarge* TX FIFO watermark too large.

*kStatus\_UART\_RxWatermarkTooLarge* RX FIFO watermark too large.

*kStatus\_UART\_FlagCannotClearManually* UART flag can't be manually cleared.

*kStatus\_UART\_Error* Error happens on UART.

*kStatus\_UART\_RxRingBufferOverrun* UART RX software ring buffer overrun.

*kStatus\_UART\_RxHardwareOverrun* UART RX receiver overrun.

*kStatus\_UART\_NoiseError* UART noise error.

*kStatus\_UART\_FramingError* UART framing error.

*kStatus\_UART\_ParityError* UART parity error.

*kStatus\_UART\_BaudrateNotSupport* Baudrate is not support in current clock source.

*kStatus\_UART\_BreakDetect* Receiver detect BREAK signal.

#### 19.2.6.2 enum uart\_data\_bits\_t

Enumerator

*kUART\_SevenDataBits* Seven data bit.

*kUART\_EightDataBits* Eight data bit.

#### 19.2.6.3 enum uart\_parity\_mode\_t

Enumerator

*kUART\_ParityDisabled* Parity disabled.

***kUART\_ParityEven*** Even error check is selected.

***kUART\_ParityOdd*** Odd error check is selected.

#### 19.2.6.4 enum uart\_stop\_bit\_count\_t

Enumerator

***kUART\_OneStopBit*** One stop bit.

***kUART\_TwoStopBit*** Two stop bits.

#### 19.2.6.5 enum \_uart\_interrupt\_enable

#### 19.2.6.6 enum \_uart\_flags

This provides constants for the UART status flags for use in the UART functions.

Enumerator

***kUART\_RxCharReadyFlag*** Rx Character Ready Flag.

***kUART\_RxErrorFlag*** Rx Error Detect Flag.

***kUART\_RxOverrunErrorFlag*** Rx Overrun Flag.

***kUART\_RxFrameErrorFlag*** Rx Frame Error Flag.

***kUART\_RxBreakDetectFlag*** Rx Break Detect Flag.

***kUART\_RxParityErrorFlag*** Rx Parity Error Flag.

***kUART\_ParityErrorFlag*** Parity Error Interrupt Flag.

***kUART\_RtsStatusFlag*** RTS\_B Pin Status Flag.

***kUART\_TxReadyFlag*** Transmitter Ready Interrupt/DMA Flag.

***kUART\_RtsDeltaFlag*** RTS Delta Flag.

***kUART\_EscapeFlag*** Escape Sequence Interrupt Flag.

***kUART\_FrameErrorFlag*** Frame Error Interrupt Flag.

***kUART\_RxReadyFlag*** Receiver Ready Interrupt/DMA Flag.

***kUART\_AgingTimerFlag*** Aging Timer Interrupt Flag.

***kUART\_DtrDeltaFlag*** DTR Delta Flag.

***kUART\_RxDsFlag*** Receiver IDLE Interrupt Flag.

***kUART\_tAIRWakeFlag*** Asynchronous IR WAKE Interrupt Flag.

***kUART\_AwakeFlag*** Asynchronous WAKE Interrupt Flag.

***kUART\_Rs485SlaveAddrMatchFlag*** RS-485 Slave Address Detected Interrupt Flag.

***kUART\_AutoBaudFlag*** Automatic Baud Rate Detect Complete Flag.

***kUART\_TxEmptyFlag*** Transmit Buffer FIFO Empty.

***kUART\_DtrFlag*** DTR edge triggered interrupt flag.

***kUART\_IdleFlag*** Idle Condition Flag.

***kUART\_AutoBaudCntStopFlag*** Auto-baud Counter Stopped Flag.

***kUART\_RiDeltaFlag*** Ring Indicator Delta Flag.

## UART Driver

*kUART\_RiFlag* Ring Indicator Input Flag.

*kUART\_IrFlag* Serial Infrared Interrupt Flag.

*kUART\_WakeFlag* Wake Flag.

*kUART\_DcdDeltaFlag* Data Carrier Detect Delta Flag.

*kUART\_DcdFlag* Data Carrier Detect Input Flag.

*kUART\_RtsFlag* RTS Edge Triggered Interrupt Flag.

*kUART\_TxCompleteFlag* Transmitter Complete Flag.

*kUART\_BreakDetectFlag* BREAK Condition Detected Flag.

*kUART\_RxOverrunFlag* Overrun Error Flag.

*kUART\_RxDataReadyFlag* Receive Data Ready Flag.

### 19.2.7 Function Documentation

#### 19.2.7.1 static void UART\_SoftwareReset ( **UART\_Type** \* *base* ) [inline], [static]

This function resets the transmit and receive state machines, all FIFOs and register USR1, USR2, UBIR, UBMR, UBRC , URXD, UTXD and UTS[6-3]

Parameters

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

#### 19.2.7.2 status\_t UART\_Init ( **UART\_Type** \* *base*, const **uart\_config\_t** \* *config*, **uint32\_t** *srcClock\_Hz* )

This function configures the UART module with user-defined settings. Call the [UART\\_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the UART.

```
* uart_config_t uartConfig;
* uartConfig.baudRate_Bps = 115200U;
* uartConfig.parityMode = kUART_ParityDisabled;
* uartConfig.dataBitsCount = kUART_EightDataBits;
* uartConfig.stopBitCount = kUART_OneStopBit;
* uartConfig.txFifoWatermark = 2;
* uartConfig.rxFifoWatermark = 1;
* uartConfig.enableAutoBaudrate = false;
* uartConfig.enableTx = true;
* uartConfig.enableRx = true;
* UART_Init(UART1, &uartConfig, 24000000U);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>config</i>	Pointer to a user-defined configuration structure.
<i>srcClock_Hz</i>	UART clock source frequency in HZ.

Return values

<i>kStatus_Success</i>	UART initialize succeed
------------------------	-------------------------

#### 19.2.7.3 void **UART\_Deinit** ( **UART\_Type** \* *base* )

This function waits for transmit to complete, disables TX and RX, and disables the UART clock.

Parameters

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

#### 19.2.7.4 void **UART\_GetDefaultConfig** ( **uart\_config\_t** \* *config* )

Gets the default configuration structure.

This function initializes the UART configuration structure to a default value. The default values are:  
: uartConfig->baudRate\_Bps = 115200U; uartConfig->parityMode = kUART\_ParityDisabled; uartConfig->dataBitsCount = kUART\_EightDataBits; uartConfig->stopBitCount = kUART\_OneStopBit; uartConfig->txFifoWatermark = 2; uartConfig->rxFifoWatermark = 1; uartConfig->enableAutoBaudrate = flase; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

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

#### 19.2.7.5 **status\_t** **UART\_SetBaudRate** ( **UART\_Type** \* *base*, **uint32\_t** *baudRate\_Bps*, **uint32\_t** *srcClock\_Hz* )

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the **UART\_Init**.

```
*   UART_SetBaudRate(UART1, 115200U, 20000000U);
*
```

## UART Driver

Parameters

<i>base</i>	UART peripheral base address.
<i>baudRate_Bps</i>	UART baudrate to be set.
<i>srcClock_Hz</i>	UART clock source frequency in Hz.

Return values

<i>kStatus_UART_Baudrate-NotSupport</i>	Baudrate is not support in the current clock source.
<i>kStatus_Success</i>	Set baudrate succeeded.

### 19.2.7.6 static void UART\_Enable ( **UART\_Type** \* *base* ) [inline], [static]

Parameters

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

### 19.2.7.7 static void UART\_Disable ( **UART\_Type** \* *base* ) [inline], [static]

Parameters

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

### 19.2.7.8 bool UART\_GetStatusFlag ( **UART\_Type** \* *base*, **uint32\_t** *flag* )

The available status flag can be select from `uart_status_flag_t` enumeration.

Parameters

<i>base</i>	UART base pointer.
<i>flag</i>	Status flag to check.

Return values

<i>current</i>	state of corresponding status flag.
----------------	-------------------------------------

### 19.2.7.9 void UART\_ClearStatusFlag ( **UART\_Type** \* *base*, **uint32\_t** *flag* )

The available status flag can be select from `uart_status_flag_t` enumeration.

Parameters

<i>base</i>	UART base pointer.
<i>flag</i>	Status flag to clear.

### 19.2.7.10 void UART\_EnableInterrupts ( **UART\_Type** \* *base*, **uint32\_t** *mask* )

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). For example, to enable TX empty interrupt and RX data ready interrupt, do the following.

```
*     UART_EnableInterrupts(UART1, kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The interrupts to enable. Logical OR of <a href="#">_uart_interrupt_enable</a> .

### 19.2.7.11 void UART\_DisableInterrupts ( **UART\_Type** \* *base*, **uint32\_t** *mask* )

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). For example, to disable TX empty interrupt and RX data ready interrupt do the following.

```
*     UART_DisableInterrupts(UART1, kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
*
```

Parameters

## UART Driver

<i>base</i>	UART peripheral base address.
<i>mask</i>	The interrupts to disable. Logical OR of <a href="#">_uart_interrupt_enable</a> .

### 19.2.7.12 **uint32\_t UART\_GetEnabledInterrupts ( UART\_Type \* *base* )**

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_uart\\_interrupt\\_enable](#). To check a specific interrupt enable status, compare the return value with enumerators in [\\_uart\\_interrupt\\_enable](#). For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = UART\_GetEnabledInterrupts\(UART1\);  
*  
*     if (kUART_TxEmptyEnable & enabledInterrupts)  
*     {  
*         ...  
*     }  
*
```

Parameters

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

Returns

UART interrupt flags which are logical OR of the enumerators in [\\_uart\\_interrupt\\_enable](#).

### 19.2.7.13 **static void UART\_EnableTx ( UART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the UART transmitter.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 19.2.7.14 **static void UART\_EnableRx ( UART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the UART receiver.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

#### 19.2.7.15 static void UART\_WriteByte ( **UART\_Type** \* *base*, **uint8\_t** *data* ) [inline], [static]

This function is used to write data to transmitter register. 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>	UART peripheral base address.
<i>data</i>	Data write to the TX register.

#### 19.2.7.16 static uint8\_t UART\_ReadByte ( **UART\_Type** \* *base* ) [inline], [static]

This function is used to read data from receiver register. 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>	UART peripheral base address.
-------------	-------------------------------

Returns

Data read from data register.

#### 19.2.7.17 void UART\_WriteBlocking ( **UART\_Type** \* *base*, **const uint8\_t** \* *data*, **size\_t** *length* )

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Note

This function does not check whether all data is sent out to the bus. Before disabling the TX, check kUART\_TransmissionCompleteFlag to ensure that the TX is finished.

## UART Driver

Parameters

<i>base</i>	UART peripheral base address.
<i>data</i>	Start address of the data to write.
<i>length</i>	Size of the data to write.

### 19.2.7.18 **status\_t UART\_ReadBlocking ( UART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )**

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the TX register.

Parameters

<i>base</i>	UART peripheral base address.
<i>data</i>	Start address of the buffer to store the received data.
<i>length</i>	Size of the buffer.

Return values

<i>kStatus_UART_Rx-HardwareOverrun</i>	Receiver overrun occurred while receiving data.
<i>kStatus_UART_Noise-Error</i>	A noise error occurred while receiving data.
<i>kStatus_UART_Framing-Error</i>	A framing error occurred while receiving data.
<i>kStatus_UART_Parity-Error</i>	A parity error occurred while receiving data.
<i>kStatus_Success</i>	Successfully received all data.

### 19.2.7.19 **void UART\_TransferCreateHandle ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uart\_transfer\_callback\_t *callback*, void \* *userData* )**

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

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

**19.2.7.20 void UART\_TransferStartRingBuffer ( *UART\_Type* \* *base*, *uart\_handle\_t* \* *handle*, *uint8\_t* \* *ringBuffer*, *size\_t* *ringBufferSize* )**

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the [UART\\_TransferReceiveNonBlocking\(\)](#) API. If data is already received in the ring buffer, the user can get the received data from the ring buffer directly.

## Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if *ringBufferSize* is 32, only 31 bytes are used for saving data.

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>ringBuffer</i>	Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
<i>ringBufferSize</i>	Size of the ring buffer.

**19.2.7.21 void UART\_TransferStopRingBuffer ( *UART\_Type* \* *base*, *uart\_handle\_t* \* *handle* )**

This function aborts the background transfer and uninstalls the ring buffer.

## Parameters

## UART Driver

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

### 19.2.7.22 size\_t **UART\_TransferGetRxRingBufferLength ( uart\_handle\_t \* handle )**

Parameters

<i>handle</i>	UART handle pointer.
---------------	----------------------

Returns

Length of received data in RX ring buffer.

### 19.2.7.23 status\_t **UART\_TransferSendNonBlocking ( UART\_Type \* base, uart\_handle\_t \* handle, uart\_transfer\_t \* xfer )**

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the [kStatus\\_UART\\_TxIdle](#) as status parameter.

Note

The [kStatus\\_UART\\_TxIdle](#) is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the [kUART\\_TxTransmissionCompleteFlag](#) to ensure that the TX is finished.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure. See <a href="#">uart_transfer_t</a> .

Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_UART_TxBusy</i>	Previous transmission still not finished; data not all written to TX register yet.
<i>kStatus_InvalidArgument</i>	Invalid argument.

#### 19.2.7.24 void **UART\_TransferAbortSend** ( **UART\_Type \* base**, **uart\_handle\_t \* handle** )

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Parameters

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

#### 19.2.7.25 status\_t **UART\_TransferGetSendCount** ( **UART\_Type \* base**, **uart\_handle\_t \* handle**, **uint32\_t \* count** )

This function gets the number of bytes written to the UART TX register by using the interrupt method.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Send bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	The parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

#### 19.2.7.26 status\_t **UART\_TransferReceiveNonBlocking** ( **UART\_Type \* base**, **uart\_handle\_t \* handle**, **uart\_transfer\_t \* xfer**, **size\_t \* receivedBytes** )

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring

## UART Driver

buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter [k\\_Status\\_UART\\_RxIdle](#). For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the *xfer->data* and this function returns with the parameter *receivedBytes* set to 5. For the left 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the UART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the *xfer->data*. When all data is received, the upper layer is notified.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure, see <a href="#">uart_transfer_t</a> .
<i>receivedBytes</i>	Bytes received from the ring buffer directly.

Return values

<i>kStatus_Success</i>	Successfully queue the transfer into transmit queue.
<i>kStatus_UART_RxBusy</i>	Previous receive request is not finished.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 19.2.7.27 void **UART\_TransferAbortReceive** ( **UART\_Type \* base**, **uart\_handle\_t \* handle** )

This function aborts the interrupt-driven data receiving. The user can get the *remainBytes* to know how many bytes are not received yet.

Parameters

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

### 19.2.7.28 status\_t **UART\_TransferGetReceiveCount** ( **UART\_Type \* base**, **uart\_handle\_t \* handle**, **uint32\_t \* count** )

This function gets the number of bytes that have been received.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferIn-Progress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

### 19.2.7.29 void **UART\_TransferHandleIRQ** ( **UART\_Type \* base, uart\_handle\_t \* handle** )

This function handles the UART transmit and receive IRQ request.

Parameters

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

### 19.2.7.30 static void **UART\_EnableTxDMA** ( **UART\_Type \* base, bool enable** ) [**inline**], [**static**]

This function enables or disables the transmit request when the transmitter has one or more slots available in the TxFIFO. The fill level in the TxFIFO that generates the DMA request is controlled by the TXTL bits.

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 19.2.7.31 static void **UART\_EnableRxDMA** ( **UART\_Type \* base, bool enable** ) [**inline**], [**static**]

This function enables or disables the receive request when the receiver has data in the RxFIFO. The fill level in the RxFIFO at which a DMA request is generated is controlled by the RXTL bits .

## UART Driver

Parameters

<i>base</i>	UART peripheral base address.
<i>enable</i>	True to enable, false to disable.

### 19.2.7.32 static void UART\_SetTxFifoWatermark ( **UART\_Type** \* *base*, **uint8\_t** *watermark* ) [inline], [static]

A maskable interrupt is generated whenever the data level in the Tx FIFO falls below the Tx FIFO watermark.

Parameters

<i>base</i>	UART base pointer.
<i>watermark</i>	The Tx FIFO watermark.

### 19.2.7.33 static void UART\_SetRxFifoWatermark ( **UART\_Type** \* *base*, **uint8\_t** *watermark* ) [inline], [static]

A maskable interrupt is generated whenever the data level in the Rx FIFO reaches the Rx FIFO watermark.

Parameters

<i>base</i>	UART base pointer.
<i>watermark</i>	The Rx FIFO watermark.

### 19.2.7.34 static void UART\_EnableAutoBaudRate ( **UART\_Type** \* *base*, **bool** *enable* ) [inline], [static]

Parameters

<i>base</i>	UART base pointer.
<i>enable</i>	Enable/Disable Automatic Baud Rate Detection feature. <ul style="list-style-type: none"><li>• true: Enable Automatic Baud Rate Detection feature.</li><li>• false: Disable Automatic Baud Rate Detection feature.</li></ul>

19.2.7.35 **static bool UART\_IsAutoBaudRateComplete ( UART\_Type \* *base* ) [inline], [static]**

## UART Driver

Parameters

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

Returns

- true: Automatic baud rate detection has finished.
  - false: Automatic baud rate detection has not finished.

## 19.2.8 Variable Documentation

- 19.2.8.1 `uint32_t uart_config_t::baudRate_Bps`
- 19.2.8.2 `uart_parity_mode_t uart_config_t::parityMode`
- 19.2.8.3 `uart_stop_bit_count_t uart_config_t::stopBitCount`
- 19.2.8.4 `uint8_t* uart_transfer_t::data`
- 19.2.8.5 `size_t uart_transfer_t::dataSize`
- 19.2.8.6 `uint8_t* volatile uart_handle_t::txData`
- 19.2.8.7 `volatile size_t uart_handle_t::txDataSize`
- 19.2.8.8 `size_t uart_handle_t::txDataSizeAll`
- 19.2.8.9 `uint8_t* volatile uart_handle_t::rxData`
- 19.2.8.10 `volatile size_t uart_handle_t::rxDataSize`
- 19.2.8.11 `size_t uart_handle_t::rxDataSizeAll`
- 19.2.8.12 `uint8_t* uart_handle_t::rxRingBuffer`
- 19.2.8.13 `size_t uart_handle_t::rxRingBufferSize`
- 19.2.8.14 `volatile uint16_t uart_handle_t::rxRingBufferHead`
- 19.2.8.15 `volatile uint16_t uart_handle_t::rxRingBufferTail`
- 19.2.8.16 `uart_transfer_callback_t uart_handle_t::callback`
- 19.2.8.17 `void* uart_handle_t::userData`
- 19.2.8.18 `volatile uint8_t uart_handle_t::txState`

### 19.3 UART FreeRTOS Driver

#### 19.3.1 Overview

#### Data Structures

- struct `uart_rtos_config_t`  
*UART configuration structure.* [More...](#)

#### UART RTOS Operation

- int `UART_RTOS_Init` (`uart_rtos_handle_t *handle`, `uart_handle_t *t_handle`, const `uart_rtos_config_t *cfg`)  
*Initializes a UART instance for operation in RTOS.*
- int `UART_RTOS_Deinit` (`uart_rtos_handle_t *handle`)  
*Deinitializes a UART instance for operation.*

#### UART transactional Operation

- int `UART_RTOS_Send` (`uart_rtos_handle_t *handle`, const `uint8_t *buffer`, `uint32_t length`)  
*Sends data in the background.*
- int `UART_RTOS_Receive` (`uart_rtos_handle_t *handle`, `uint8_t *buffer`, `uint32_t length`, `size_t *received`)  
*Receives data.*

#### 19.3.2 Data Structure Documentation

##### 19.3.2.1 struct `uart_rtos_config_t`

#### Data Fields

- `UART_Type * base`  
*UART base address.*
- `uint32_t srclk`  
*UART source clock in Hz.*
- `uint32_t baudrate`  
*Desired communication speed.*
- `uart_parity_mode_t parity`  
*Parity setting.*
- `uart_stop_bit_count_t stopbits`  
*Number of stop bits to use.*
- `uint8_t * buffer`  
*Buffer for background reception.*
- `uint32_t buffer_size`  
*Size of buffer for background reception.*

### 19.3.3 Function Documentation

19.3.3.1 `int UART_RTOS_Init( uart_rtos_handle_t * handle, uart_handle_t * t_handle,  
const uart_rtos_config_t * cfg )`

## UART FreeRTOS Driver

Parameters

<i>handle</i>	The RTOS UART handle, the pointer to an allocated space for RTOS context.
<i>t_handle</i>	The pointer to the allocated space to store the transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the UART after initialization.

Returns

0 succeed; otherwise fail.

### 19.3.3.2 int **UART\_RTOS\_Deinit** ( *uart\_rtos\_handle\_t \* handle* )

This function deinitializes the UART module, sets all register values to reset value, and frees the resources.

Parameters

<i>handle</i>	The RTOS UART handle.
---------------	-----------------------

### 19.3.3.3 int **UART\_RTOS\_Send** ( *uart\_rtos\_handle\_t \* handle, const uint8\_t \* buffer, uint32\_t length* )

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

<i>handle</i>	The RTOS UART handle.
<i>buffer</i>	The pointer to the buffer to send.
<i>length</i>	The number of bytes to send.

### 19.3.3.4 int **UART\_RTOS\_Receive** ( *uart\_rtos\_handle\_t \* handle, uint8\_t \* buffer, uint32\_t length, size\_t \* received* )

This function receives data from UART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

## Parameters

<i>handle</i>	The RTOS UART handle.
<i>buffer</i>	The pointer to the buffer to write received data.
<i>length</i>	The number of bytes to receive.
<i>received</i>	The pointer to a variable of size_t where the number of received data is filled.



# Chapter 20

## MMDC: Multi Mode DDR Controller Driver

### 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Multi Mode DDR Controller block of MCUXpresso SDK devices.

The MMDC is configurable high performance and optimized DDR controller that support LPDDR2 x16 memory type. The MMDC driver provide function API to config the MMDC core and PHY register.

### 20.2 Typical use case

This example shows how to use the MMDC driver to initialize the external 16 bit DDR device. Initialize the MMDC controller and run the initialization sequence. The external DDR device is initialized and the DDR read and write is available.

Initialize the LPDDR2 Controller and external device.

```
mmdc_device_timing_t timing =
{
    .tCKE_Clocks = 3U,
    .tCKSRE_Clocks = 2U,
    .tCKSRX_Clocks = 2U,
    .tRFC_Clocks = 40U,
    .tXSR_Clocks = 43U,
    .tXP_Clocks = 3U,
    .tFAW_Clocks = 16U,
    .tCL_Clocks = 3U,
    .tMRD_Clocks = 3U,
    .tRAS_Clocks = 13U,
    .tWR_Clocks = 5U,
    .tRTP_Clocks = 3U,
    .tWTR_Clocks = 4U,
    .tRRD_Clocks = 4U,
    .ralat_Clocks = 5U,
    .walat_Clocks = 0U,
    .tRC_Clocks = 19U,
    .tRCD_Clocks = 6U,
    .tRP_Clocks = 6U,
    .tRPA_Clocks = 7U,
    .tcwl_Clocks = 1U,
    .tRSTtoCKE_Clocks = 14U,
    .txpr_Clocks = 159U,
    .trtwsame_Clocks = 2U,
    .twtrdiff_Clocks = 3U,
    .twtwdiff_Clocks = 3U,
    .trtrdiff_Clocks = 2U,
    .tdai_Clocks = 3001U,
};

mmdc_fine_tuning_config_t tuning =
{
    .caDelay = 0x47U,
    .rdQOffset0 = 0x33333333U,
```

## Typical use case

```
.rDQOffset1 = 0x33333333U,
.wDQOffset0 = 0xf3333333U,
.wDQOffset1 = 0xf3333333U,
.rDQDuty0 = kMMDC_DutyHighPercent50,
.rDQDuty1 = kMMDC_DutyHighPercent50,
.ddrCKDutyCtl0 = kMMDC_DutyHighPercent50,
.ddrCKDutyCtl1 = kMMDC_DutyHighPercent50,
.wDQDuty0 = kMMDC_DutyHighPercent50,
.wDQDuty1 = kMMDC_DutyHighPercent50,
};

mmdc_zq_config_t zqCal =
{
    .mode = kMMDC_ZQCaltoIODeviceLongShort,
    .earlyCompTimer = 21U,
    .hwZQFreq = kMMDC_ZQCalFreq1ms,
    .cmpOutSample = 15U,
    .tZQCl_Clocks = 144U,
    .tZQCs_Clocks = 112U,
    .tZQInit_Clocks = 400U,
    .cmpOutSample = 7,
    .hwPullDownOffset = 0U,
    .hwPullUpOffset = 0U,
};

mmdc_config_t userConfig;
mmdc_zq_config_t zqCal;
mmdc_read_calibration_config_t readCal;
mmdc_write_calibration_config_t writeCal;
mmdc_device_config_t config;
mmdc_auto_refresh_t autoRefresh;

readCal.mode = kMMDC_CalWithMPR;
readCal.readDelay0 = 0x40U;
readCal.readDelay1 = 0x40U;

writeCal.mode = kMMDC_CalWithPreDefine;
writeCal.writeDelay0 = 0x40U;
writeCal.writeDelay1 = 0x40U;

config.MR1 = 0x82U;
config.MR2 = 0x01U;
config.MR3 = 0x01U;

autoRefresh.refreshCnt = 0;
autoRefresh.refreshRate = 4;
autoRefresh.refreshTrigSrc = kMMDC_RefreshTrigBy64K;

/*
 * config->bankInterleave = true;
 * config->secondDDRClock = true;
 * config->enableOnlyCS0 = true;
 * config->devType = kMMDC_DDR3;
 * config->devSize = 0x40000000U;
 * config->devBank = kMMDC_Bank8;
 * config->rowWidth = kMMDC_Row16Bits;
 * config->colWidth = kMMDC_Col10Bits;
 * config->burstLen = kMMDC_BurstLen8;
 * config->odtByte1Config = kMMDC_RttNom60ohm;
 * config->odtByte0Config = kMMDC_RttNom60ohm;
 * config->enableActiveReadOdt = false;
 * config->enableInactiveReadOdt = true;
 * config->enableActiveWriteOdt = true;
 * config->enableInactiveWriteOdt = true;
 */
MMDC_GetDefaultConfig(&userConfig);
userConfig.devType = kMMDC_LPDDR2_S4;
userConfig.devSize = EXAMPLE_DDR_SIZE;
```

```

userConfig.rowWidth = EXAMPLE_DDR_ROW_WIDTH;
userConfig.colWidth = EXAMPLE_DDR_COL_WIDTH;
userConfig.burstLen = EXAMPLE_DDR_BURST_LEN;
userConfig.timing = &timing;
userConfig.deviceConfig[0] = &config;
userConfig.readCalibration[0] = &readCal;
userConfig.writeCalibration[0] = &writeCal;
userConfig.tuning = &tuning;
userConfig.autoRefresh = &autoRefresh;
userConfig.zqCalibration = &zqCal;

/* MMDC Initialization. */
if (MMDC_Init(EXAMPLE_MMDC, &userConfig) != kStatus_Success)
{
    PRINTF("\r\n MMDC Init Failed. \r\n");
}

```

## Data Structures

- struct [mmdc\\_readDQS\\_calibration\\_config\\_t](#)  
*MMDC read DQS gating calibration configuration collection.* [More...](#)
- struct [mmdc\\_writeLeveling\\_calibration\\_config\\_t](#)  
*MMDC write leveling calibration configuration collection.* [More...](#)
- struct [mmdc\\_read\\_calibration\\_config\\_t](#)  
*MMDC read calibration configuration collection.* [More...](#)
- struct [mmdc\\_fine\\_tuning\\_config\\_t](#)  
*MMDC write calibration configuration collection.* [More...](#)
- struct [mmdc\\_odt\\_config\\_t](#)  
*MMDC odt configuration collection.* [More...](#)
- struct [mmdc\\_power\\_config\\_t](#)  
*MMDC power configuration collection.* [More...](#)
- struct [mmdc\\_zq\\_config\\_t](#)  
*MMDC ZQ configuration collection.* [More...](#)
- struct [mmdc\\_cmd\\_config\\_t](#)  
*MMDC cmd configuration collection.* [More...](#)
- struct [mmdc\\_device\\_timing\\_t](#)  
*MMDC device device timing configuration collection.* [More...](#)
- struct [mmdc\\_auto\\_refresh\\_t](#)  
*MMDC auto refresh configuration collection.* [More...](#)
- struct [mmdc\\_exaccess\\_config\\_t](#)  
*MMDC exclusive access configuration collection.* [More...](#)
- struct [mmdc\\_profiling\\_config\\_t](#)  
*MMDC module profiling configuration collection.* [More...](#)
- struct [mmdc\\_performance\\_config\\_t](#)  
*MMDC performance configuration collection.* [More...](#)
- struct [mmdc\\_device\\_config\\_t](#)  
*MMDC module relate configuration collection.* [More...](#)
- struct [mmdc\\_config\\_t](#)  
*MMDC module relate configuration collection.* [More...](#)

## Typedefs

- [typedef void\(\\* MMDC\\_SwitchFrequency \)\(MMDC\\_Type \\*, void \\*, void \\*, uint32\\_t\)](#)  
*MMDC switch frequency api prototype.*

## Typical use case

### Enumerations

- enum `_mmdc_status` {  
    kStatus\_MMDC\_ErrorDGCalibration = MAKE\_STATUS(kStatusGroup\_MMDC, 1),  
    kStatus\_MMDC\_ErrorReadCalibration = MAKE\_STATUS(kStatusGroup\_MMDC, 2),  
    kStatus\_MMDC\_ErrorWriteCalibration = MAKE\_STATUS(kStatusGroup\_MMDC, 3),  
    kStatus\_MMDC\_ErrorWriteLevelingCalibration = MAKE\_STATUS(kStatusGroup\_MMDC, 4),  
    kStatus\_MMDC\_WaitFlagTimeout = MAKE\_STATUS(kStatusGroup\_MMDC, 5) }  
    *MMDC status return codes.*
- enum `mmdc_device_type_t` {  
    kMMDC\_LPDDR2\_S4 = 0x0U,  
    kMMDC\_LPDDR2\_S2 = 0x1U,  
    kMMDC\_DDR3 = 0x2U }  
    *LPDDR2 device list.*
- enum `mmdc_device_bank_num_t` {  
    kMMDC\_Bank8 = 0U,  
    kMMDC\_Bank4 = 1U }  
    *LPDDR2 device bank number.*
- enum `mmdc_row_addr_width_t` {  
    kMMDC\_Row11Bits = 0U,  
    kMMDC\_Row12Bits = 1U,  
    kMMDC\_Row13Bits = 2U,  
    kMMDC\_Row14Bits = 3U,  
    kMMDC\_Row15Bits = 4U,  
    kMMDC\_Row16Bits = 5U }  
    *define for row addr width*
- enum `mmdc_col_addr_width_t` {  
    kMMDC\_Col9Bits = 0U,  
    kMMDC\_Col10Bits = 1U,  
    kMMDC\_Col11Bits = 2U,  
    kMMDC\_Col8Bits = 3U,  
    kMMDC\_Col12Bits = 4U }  
    *define for col addr width*
- enum `mmdc_burst_len_t` {  
    kMMDC\_BurstLen4 = 0U,  
    kMMDC\_BurstLen8,  
    kMMDC\_BurstLen16 }  
    *define for burst length*
- enum `mmdc_cmd_type_t` {  
    kMMDC\_NormalOperation = 0x0U,  
    kMMDC\_AutoRefresh = 0x2U,  
    kMMDC\_WriteModeRegister = 0x3U,  
    kMMDC\_ZQCalibration = 0x04U,  
    kMMDC\_PreChargeAll = 0x05U,  
    kMMDC\_ReadModeRegister = 0x6U }  
    *define for command type auto refresh cmd: select correct CMD\_CS before issue this cmd write mode  
register cmd: DDR2/DDR3: CMD\_CS,CMD\_BA,CMD\_ADDR\_LSB,CMD\_ADDR\_MSB LPDDR2: CM-*

*D\_CS,MR\_OP,MR\_ADDR ZQ calibration cmd: DDR2/DDR3: CMD\_CS,{CMD\_ADDR\_MSB,CMD\_A-DDR\_LSB}=0x400,or 0x0 LPDDR2: through MRW cmd PreChargeAll cmd: selecet correct CMD\_CS MRR cmd: only for LPDDR2/LPDDR3 device,must set correct CMD\_CS,MR\_ADDR*

- enum `mmdc_zq_calmode_t` {
   
kMMDC\_ZQCaltoIOHW = 0x4U,
   
kMMDC\_ZQCaltoIODeviceLong = 0x01U,
   
kMMDC\_ZQCaltoDeviceOnly = 0x02U,
   
kMMDC\_ZQCaltoIODeviceLongShort,
   
kMMDC\_ZQFinetuning = 0x0AU,
   
kMMDC\_DisZQFinetuning = 0x0BU }

*MMDC ZQ calibration type.*

- enum `mmdc_zq_calfreq_t` {
   
kMMDC\_ZQCalFreq1ms = 0x0U,
   
kMMDC\_ZQCalFreq2ms = 0x1U,
   
kMMDC\_ZQCalFreq4ms = 0x2U,
   
kMMDC\_ZQCalFreq1s = 0x6U,
   
kMMDC\_ZQCalFreq16s = 0xEU,
   
kMMDC\_ZQCalFreq32s = 0xFU }

*MMDC ZQ calibration frequency.*

- enum `mmdc_refresh_sel_t` {
   
kMMDC\_RefreshTrigBy64K = 0U,
   
kMMDC\_RefreshTrigBy32K,
   
kMMDC\_RefreshTrigDDRCycles,
   
kMMDC\_RefreshTrigNone }

*define MMDC refresh selector-select source of the clock that will trigger each refresh cycle.*

- enum `mmdc_profiling_action_t` {
   
kMMDC\_EnProfilingWithID,
   
kMMDC\_FreezeProfiling,
   
kMMDC\_CheckOverFlow }

*define MMDC profiling action define .*

- enum `mmdc_calibration_type_t` {
   
kMMDC\_CalWithPreSetValue,
   
kMMDC\_CalWithMPR,
   
kMMDC\_CalWithPreDefine }

*MMDC calibration type define .*

- enum `mmdc_calibaration_waitycycles_t` {
   
kMMDC\_Wait16DDRCycles = 0U,
   
kMMDC\_Wait32DDRCycles = 0x01U }

*define MMDC wait cycles before comparing data during calibration.*

- enum `mmdc_fine_tuning_dutycycle_t` {
   
kMMDC\_DutyHighPercent48\_5 = 0x1U,
   
kMMDC\_DutyHighPercent50 = 0x2U,
   
kMMDC\_DutyHighPercent51\_5 = 0x4U }

*define MMDC parameter fine tuning duty cyle.*

- enum `mmdc_termination_config_t` {

## Typical use case

```
kMMDC_RttNomDisabled = 0x0U,  
kMMDC_RttNom120ohm = 0x1U,  
kMMDC_RttNom60ohm = 0x2U,  
kMMDC_RttNom40ohm = 0x3U,  
kMMDC_RttNom30ohm = 0x4U,  
kMMDC_RttNom24ohm = 0x5U,  
kMMDC_RttNom20ohm = 0x6U,  
kMMDC_RttNom17ohm = 0x7U }  
    define MMDC on chip termination configurations.  
• enum _mmdc_lpddr2_derate {  
    kMMDC_NoUpdateDerate = 0U,  
    kMMDC_UpdateRefreshRate = 0x01U,  
    kMMDC_DerateTiming = 0x02U }  
        define LPDDR2 device derating type.  
• enum _mmdc_exaccess_type {  
    kMMDC_ExMonitorID0 = 0x01U,  
    kMMDC_ExMonitorID1 = 0x02U,  
    kMMDC_ExMonitorID2 = 0x04U,  
    kMMDC_ExMonitorID3 = 0x08U,  
    kMMDC_ExAccessResponse = 0x10U }  
        MMDC exclusive access config type.
```

## Driver version

- #define **FSL\_MMDC\_DRIVER\_VERSION** (MAKE\_VERSION(2U, 1U, 1U))  
*MMDC driver Version 2.1.1.*
- #define **MMDC\_TIMEOUT** (500U)  
*MMDC retry times.*
- #define **MMDC\_READ\_DQS\_FINE\_TUNING\_MASK** (0x77777777U)  
<
- #define **MMDC\_WRITE\_DQS\_FINE\_TUNING\_MASK** (0xF3333333U)  
<
- #define **MMDC\_PRE\_DEFINE\_VALUE\_DEFAULT** (0xCCU)  
<
- #define **MMDC\_MEASUREUNIT\_ERR\_FREQ** (100000000U)  
<

## Initialization and deinitialization

- void **MMDC\_GetDefaultConfig** (**mmdc\_config\_t** \*config)  
*MMDC module get the default configuration get timing/power/zq configuration.*
- status\_t **MMDC\_Init** (**MMDC\_Type** \*base, **mmdc\_config\_t** \*config)  
*MMDC module initialization function.*
- void **MMDC\_Deinit** (**MMDC\_Type** \*base)  
*MMDC module deinit function.*

## device operation

- void **MMDC\_HandleCommand** (MMDC\_Type \*base, mmdc\_cmd\_config\_t \*config)  
*MMDC module process the command,support transfer multi cmd in once function call.*
- status\_t **MMDC\_GetReadData** (MMDC\_Type \*base, uint32\_t \*data)  
*MMDC get the read data.*
- void **MMDC\_EnhancePerformance** (MMDC\_Type \*base, const mmdc\_performance\_config\_t \*config)  
*MMDC module enhance performance function.*
- void **MMDC\_EnableAutoRefresh** (MMDC\_Type \*base, mmdc\_auto\_refresh\_t \*config)  
*Enable MMDC module periodic refresh scheme config.*
- static void **MMDC\_DisableAutoRefresh** (MMDC\_Type \*base)  
*Disable MMDC module periodic refresh scheme.*
- void **MMDC\_EnablePowerSaving** (MMDC\_Type \*base, mmdc\_power\_config\_t \*config)  
*MMDC enable automatic power saving.*
- static void **MMDC\_DisablePowerSaving** (MMDC\_Type \*base)  
*uint8\_t targetCS,*
- void **MMDC\_Profiling** (MMDC\_Type \*base, mmdc\_profiling\_config\_t \*config)  
*MMDC profiling mechanism.*
- status\_t **MMDC\_LPDDR2UpdateDerate** (MMDC\_Type \*base, mmdc\_auto\_refresh\_t \*config, uint32\_t type)  
*MMDC update device refresh rate and derate timing for LPDDR2 device only.*
- status\_t **MMDC\_MonitorLPDDR2OperationTemp** (MMDC\_Type \*base, uint32\_t \*mr4)  
*MMDC device operation temp monitor function.*
- status\_t **MMDC\_ReadDQSGatingCalibration** (MMDC\_Type \*base, mmdc\_readDQS\_calibration\_config\_t \*config)  
*MMDC module read DQS gating calibration function.*
- status\_t **MMDC\_WriteLevelingCalibration** (MMDC\_Type \*base, mmdc\_writeLeveling\_calibration\_config\_t \*config)  
*MMDC module write leveling calibration function.*
- status\_t **MMDC\_WriteCalibration** (MMDC\_Type \*base, mmdc\_write\_calibration\_config\_t \*config)  
*MMDC module write calibration function.*
- status\_t **MMDC\_ReadCalibration** (MMDC\_Type \*base, mmdc\_read\_calibration\_config\_t \*config)  
*MMDC module read calibration function.*
- void **MMDC\_DoFineTuning** (MMDC\_Type \*base, mmdc\_device\_type\_t devType, mmdc\_fine\_tuning\_config\_t \*config)  
*MMDC module read calibration function.*
- void **MMDC\_SetTiming** (MMDC\_Type \*base, mmdc\_device\_type\_t devType, mmdc\_device\_timing\_t \*timing)  
*set timing parameter*
- void **MMDC\_DeviceInit** (MMDC\_Type \*base, mmdc\_device\_type\_t devType, uint8\_t targetCS, mmdc\_device\_config\_t \*devConfig)  
*Initialize MMDC controlled device.*
- status\_t **MMDC\_EnterConfigurationMode** (MMDC\_Type \*base, bool enable)  
*MMDC module enter/exit configuration mode function.*
- void **MMDC\_DoZQCalibration** (MMDC\_Type \*base, mmdc\_device\_type\_t devType, mmdc\_zq\_config\_t \*zqCal)  
*MMDC do ZQ calibration function.*
- status\_t **MMDC\_EnableLowPowerMode** (MMDC\_Type \*base, bool enable)

## Data Structure Documentation

- **MMDC\_enable/disable low power mode** Once enable device will enter self-refresh mode.
- status\_t **MMDC\_EnableDVFSMode** (MMDC\_Type \*base, bool enable)  
*MMDC enable/disable dynamic frequency change mode Once enable device will enter self-refresh mode.*
- void **MMDC\_Reset** (MMDC\_Type \*base)  
*MMDC module reset function when you call this function will reset all internal register user need to module init function bue do not need to do device init.*
- static void **MMDC\_SwitchDeviceFrequency** (MMDC\_Type \*base, void \*ccm, void \*iomux, uint32\_t param, uint32\_t codeAddr)  
*define the mmdc switch frequency.*

## debug

- static void **MMDC\_EnableSBS** (MMDC\_Type \*base, bool enable)  
*MMDC enable/disable the SBS-step by step debug feature.*
- static void **MMDC\_TriggerSBS** (MMDC\_Type \*base)  
*MMDC trigger the MMDC dispatch the one pending request to device.*
- static uint32\_t **MMDC\_GetAXIAddrBySBS** (MMDC\_Type \*base)  
*MMDC get AXI ddr which was dispatched by MMDC in SBS mode.*
- static uint32\_t **MMDC\_GetAXIAttributeBySBS** (MMDC\_Type \*base)  
*MMDC get AXI attribute which was dispatched by MMDC in SBS mode.*
- static void **MMDC\_EnableProfiling** (MMDC\_Type \*base, bool enable)  
*MMDC enable/disable profiling feature.*
- static void **MMDC\_ResumeProfiling** (MMDC\_Type \*base)  
*MMDC resume profiling.*
- static void **MMDC\_ResetProfiling** (MMDC\_Type \*base)  
*MMDC reset profiling.*
- void **MMDC\_ExclusiveAccess** (MMDC\_Type \*base, mmdc\_exaccess\_config\_t \*config, uint32\_t type)  
*MMDC exclusive access config function, config the monitor ID and response.*

## 20.3 Data Structure Documentation

### 20.3.1 struct mmdc\_readDQS\_calibration\_config\_t

#### Data Fields

- mmdc\_calibration\_type\_t mode  
*select calibration mode.*
- mmdc\_calibration\_waitcycles\_t waitCycles  
*MMDC wait cycles before comparing sample data.*
- uint8\_t dqsGatingHalfDelay0  
*Read DQS gating half cycles delay count for Byte0.*
- uint8\_t dqsGatingAbsDelay0  
*Absolute read DQS gating delay offset for Byte0, So the total read DQS gating delay is (dqsGatingHalfDelay0)\*0.5\*cycle + (dqsGatingAbsDelay0)\*1/256\*cycle.*
- uint8\_t dqsGatingHalfDelay1  
*Read DQS gating half cycles delay count for Byte1.*
- uint8\_t dqsGatingAbsDelay1  
*Absolute read DQS gating delay offset for Byte1, So the total read DQS gating delay is (dqsGatingHalfDelay1)\*0.5\*cycle + (dqsGatingAbsDelay1)\*1/256\*cycle.*

- `uint8_t readDelay0`

*When using hardware calibration(MPR/Predefined mode), user should input RD\_DL\_ABS\_OFFSET to place read DQS inside the read DQ window.*

- `uint8_t readDelay1`

*When using hardware calibration(MPR/Predefined mode), user should input RD\_DL\_ABS\_OFFSET to place read DQS inside the read DQ window.*

### 20.3.1.0.0.32 Field Documentation

#### 20.3.1.0.0.32.1 `mmdc_calibration_type_t mmdc_readDQS_calibration_config_t::mode`

#### 20.3.1.0.0.32.2 `mmdc_calibaration_waitcycles_t mmdc_readDQS_calibration_config_t::waitCycles`

#### 20.3.1.0.0.32.3 `uint8_t mmdc_readDQS_calibration_config_t::dqsGatingHalfDelay0`

#### 20.3.1.0.0.32.4 `uint8_t mmdc_readDQS_calibration_config_t::dqsGatingAbsDelay0`

#### 20.3.1.0.0.32.5 `uint8_t mmdc_readDQS_calibration_config_t::dqsGatingHalfDelay1`

#### 20.3.1.0.0.32.6 `uint8_t mmdc_readDQS_calibration_config_t::dqsGatingAbsDelay1`

#### 20.3.1.0.0.32.7 `uint8_t mmdc_readDQS_calibration_config_t::readDelay0`

#### 20.3.1.0.0.32.8 `uint8_t mmdc_readDQS_calibration_config_t::readDelay1`

### 20.3.2 `struct mmdc_writeLeveling_calibration_config_t`

#### Data Fields

- `mmdc_calibration_type_t mode`

*select calibration mode.*

- `uint8_t wLevelingOneDelay0`

*Write leveling one cycles delay count for Byte0.*

- `uint8_t wLevelingHalfDelay0`

*Write leveling half cycles delay count for Byte0.*

- `uint8_t wLevelingAbsDelay0`

Absolute Write leveling delay offset for Byte0, So the total delay is the  $(wLevelingAbsDelay0/256*cycle) + (wLevelingHalfDelay0*halfcycle) + (wLevelingOneDelay0*cycle)$ .

- `uint8_t wLevelingOneDelay1`

*Write leveling one cycles delay count for Byte1.*

- `uint8_t wLevelingHalfDelay1`

*Write leveling half cycles delay count for Byte1.*

- `uint8_t wLevelingAbsDelay1`

Absolute Write leveling delay offset for Byte1, So the total delay is the  $(wLevelingAbsDelay1/256*cycle) + (wLevelingHalfDelay1*halfcycle) + (wLevelingOneDelay1*cycle)$ .

## Data Structure Documentation

### 20.3.2.0.0.33 Field Documentation

20.3.2.0.0.33.1 `mmdc_calibration_type_t mmdc_writeLeveling_calibration_config_t::mode`

20.3.2.0.0.33.2 `uint8_t mmdc_writeLeveling_calibration_config_t::wLevelingAbsDelay0`

20.3.2.0.0.33.3 `uint8_t mmdc_writeLeveling_calibration_config_t::wLevelingAbsDelay1`

### 20.3.3 `struct mmdc_read_calibration_config_t`

#### Data Fields

- `mmdc_calibration_type_t mode`  
*select calibration mode.*
- `uint8_t readDelay0`  
*delay between read DQS strobe and read data of Byte0, RD\_DL\_ABS\_OFFSET0.*
- `uint8_t readDelay1`  
*delay between read DQS strobe and read data of Byte1, RD\_DL\_ABS\_OFFSET1.*

### 20.3.3.0.0.34 Field Documentation

20.3.3.0.0.34.1 `mmdc_calibration_type_t mmdc_read_calibration_config_t::mode`

20.3.3.0.0.34.2 `uint8_t mmdc_read_calibration_config_t::readDelay0`

The delay of the delay-line would be (RD\_DL\_ABS\_OFFSET0 / 256) \* MMDC AXIclock (fast clock). when using hardware calibration(MPR/Predefined mode), user should input RD\_DL\_ABS\_OFFSET0 to place read DQS inside the read DQ window.

20.3.3.0.0.34.3 `uint8_t mmdc_read_calibration_config_t::readDelay1`

The delay of the delay-line would be (RD\_DL\_ABS\_OFFSET1 / 256) \* MMDC AXIclock (fast clock). when using hardware calibration(MPR/Predefined mode), user should input RD\_DL\_ABS\_OFFSET1 to place read DQS inside the read DQ window.

### 20.3.4 `struct mmdc_fine_tuning_config_t`

#### Data Fields

- `uint32_t rDQOffset0`  
*fine-tuning adjustment to every bit in the read DQ byte0 relative to the read DQS, max dealy units can be add is 7*
- `uint32_t rDQOffset1`  
*fine-tuning adjustment to every bit in the read DQ byte1 relative to the read DQS, max dealy units can be add is 7*
- `uint32_t wDQOffset0`  
*fine-tuning adjustment to every bit in the write DQ byte0 relative to the write DQS, max dealy units can be*

- `uint32_t wDQOffset1`  
*fine-tuning adjustment to every bit in the write DQ byte1 relative to the write DQS, max dealy units can be add is 7*
- `uint32_t caDelay`  
*CA delay line fine tuning parameter.*
- `mmdc_fine_tuning_dutycycle_t rDQDuty0`  
*Read DQS duty cycle fine tuning control of Byte1.*
- `mmdc_fine_tuning_dutycycle_t rDQDuty1`  
*Read DQS duty cycle fine tuning control of Byte0.*
- `mmdc_fine_tuning_dutycycle_t ddrCKDutyCtl0`  
*Primary duty cycle fine tuning control of DDR clock.*
- `mmdc_fine_tuning_dutycycle_t ddrCKDutyCtl1`  
*Secondary duty cycle fine tuning control of DDR clock.*
- `mmdc_fine_tuning_dutycycle_t wDQDuty0`  
*Write DQS duty cycle fine tuning control of Byte0.*
- `mmdc_fine_tuning_dutycycle_t wDQDuty1`  
*Write DQS duty cycle fine tuning control of Byte1.*

#### 20.3.4.0.0.35 Field Documentation

##### 20.3.4.0.0.35.1 `uint32_t mmdc_fine_tuning_config_t::caDelay`

#### 20.3.5 `struct mmdc_odt_config_t`

#### Data Fields

- `mmdc_termination_config_t odtByte0Config`  
*On chip ODT byte1 resistor.*
- `bool enableActiveReadOdt`  
*On chip ODT byte0 resistor.*
- `bool enableInactiveReadOdt`  
*Active read CS ODT enable.*
- `bool enableActiveWriteOdt`  
*Inactive read CS ODT enable.*
- `bool enableInactiveWriteOdt`  
*Active write CS ODT enable.*

## Data Structure Documentation

### 20.3.5.0.0.36 Field Documentation

20.3.5.0.0.36.1 `mmdc_termination_config_t mmdc_odt_config_t::odtByte0Config`

20.3.5.0.0.36.2 `bool mmdc_odt_config_t::enableActiveReadOdt`

20.3.5.0.0.36.3 `bool mmdc_odt_config_t::enableInactiveReadOdt`

20.3.5.0.0.36.4 `bool mmdc_odt_config_t::enableActiveWriteOdt`

20.3.5.0.0.36.5 `bool mmdc_odt_config_t::enableInactiveWriteOdt`

### 20.3.6 struct `mmdc_power_config_t`

#### Data Fields

- `bool wIdle`  
*get write request buffer Idle status*
- `bool rIdle`  
*get read request buffer Idle status*
- `bool isInAutoPS`  
*indicate mmdc if in a automatic power saving mode*
- `uint8_t idleClockToPS`  
*define the idle clock which device will automatically enter auto self-refresh mode ,default is 1024 clock cycles, max is 16320 cycles,calucate formula is idleClockToPS \* 64 = idle clock note: idleClockToPS = 0 is forbidden*
- `uint8_t idleClockToPrecharge0`  
*define the idle clock which device will automatically precharged.*
- `uint8_t idleClockToPD0`  
*define the idle clock which device will enter power down, default is disable ,max clock is 32768 clocks, calucate formula idleClockToPD \*16 = idle clock*
- `uint8_t idleClockToPrecharge1`  
*define the idle clock which device will automatically precharged.*
- `uint8_t idleClockToPD1`  
*define the idle clock which device will enter power down, default is disable ,max clock is 32768 clocks, calucate formula idleClockToPD \*16 = idle clock*

### 20.3.6.0.0.37 Field Documentation

20.3.6.0.0.37.1 `uint8_t mmdc_power_config_t::idleClockToPrecharge0`

default is disable ,max clock is 128 clocks, calucate formula  $2^{\wedge}$ idleClockToPrecharge = idle clock

20.3.6.0.0.37.2 `uint8_t mmdc_power_config_t::idleClockToPrecharge1`

default is disable ,max clock is 128 clocks, calucate formula  $2^{\wedge}$ idleClockToPrecharge = idle clock

### 20.3.7 struct mmdc\_zq\_config\_t

#### Data Fields

- **mmdc\_zq\_calmode\_t mode**  
*zq calibration mode.*
- **uint8\_t earlyCompTimer**  
*this field define the interval between the warming up of the comp of the ultra cal pad and the begining of the ZQ cal process with pads*
- **uint16\_t tZQCl\_Clocks**  
*This is the period of time that the MMDC has to wait after sending a short ZQ calibration and before sending other commands,max value 112 cycles,see RM for more detail ,lpddr2 device default is 360ns.*
- **uint16\_t tZQCcs\_Clocks**  
*This is the period of time that the MMDC has to wait after sending a long ZQ calibration and before sending other commands.*
- **uint16\_t tZQInit\_Clocks**  
*This is the period of time that the MMDC has to wait after sending a init ZQ calibration and before sending other commands.lpddr2 device default is 1us.*
- **mmdc\_zq\_calfreq\_t hwZQFreq**  
*ZQ periodic calibration freq.*
- **uint8\_t cmpOutSample**  
*define the amount of cycle between driving the ZQ signal to pad and till sampling the cmp enable output*
- **uint8\_t hwPullDownOffset**  
*define ZQ hardware pull down offset, used for fine tuning*
- **uint8\_t hwPullUpOffset**  
*define ZQ hardware pull up offset, used for fine tuning*

#### 20.3.7.0.0.38 Field Documentation

##### 20.3.7.0.0.38.1 mmdc\_zq\_calmode\_t mmdc\_zq\_config\_t::mode

### 20.3.8 struct mmdc\_cmd\_config\_t

#### Data Fields

- **uint8\_t argMsb**  
*define the CMD\_ADDR\_MSB\_MR\_OP, for lpddr2 device this field is mode register oprand*
- **uint8\_t argLsb**  
*define the CMD\_ADDR\_LSB\_MR\_ADDR,for lpddr2 device this field is mode register addr*
- **uint8\_t bankAddr**  
*define the bank address, this field not relate with lpddr2 device*
- **uint8\_t targetCS**  
*select which CS to drive low.*
- **mmdc\_cmd\_type\_t cmd**  
*define the cmd to be send*

## Data Structure Documentation

### 20.3.8.0.0.39 Field Documentation

#### 20.3.8.0.0.39.1 uint8\_t mmdc\_cmd\_config\_t::targetCS

### 20.3.9 struct mmdc\_device\_timing\_t

clocks is ddr clock,(a+b), a is the value write to reigster,b is offset

## Data Fields

- uint8\_t **tRFC\_Clocks**  
Refresh cmd to active or refresh cmd time  
*default is (0x32+1) clocks,max is (255+1) clocks*
- uint8\_t **tCKSRX\_Clocks**  
Valid clock before self-refresh exit,self-refresh timing  
*default is 2 clocks,max is 7 clocks*
- uint8\_t **tCKSRE\_Clocks**  
Valid clock after self-refresh entry,self-refresh timing  
*default is 2 clocks,max is 7 clocks*
- uint8\_t **tXSR\_Clocks**  
exit self refresh to a valid cmd,self-refresh timing  
*min value should set to 0x16,represent 23 clocks,max is 256 clocks*
- uint8\_t **tCKE\_Clocks**  
*CKE minimum pulse width,default is (3+1) clocks,max is (7+1) clocks.*
- uint8\_t **tCL\_Clocks**  
*CAS read latency,default is(3+3) clocks, max is (8+3) clocks.*
- uint8\_t **tCWL\_Clocks**  
*CAS write latency, default is (3+1)clocks, max is (6+1) clocks.*
- uint8\_t **ralat\_Clocks**  
*define write additional latency in misc, default is disable, max is (7+2)clocks*
- uint8\_t **walat\_Clocks**  
*define read additional latency in misc, default is disable, max is 3 clocks*
- uint8\_t **tFAW\_Clocks**  
*Four bank active window,all bank,default is (6+1) clocks,max is (31+1)clocks.*
- uint8\_t **tRAS\_Clocks**  
*row active time,Active to Precharge cmd period,same bank,default is (9+1) clocks ,max is (30+1)clocks*
- uint8\_t **tRC\_Clocks**  
*Active to active or refresh cmd period,default is (0+1)clocks,max is (62+1)clocks.*
- uint8\_t **tRCD\_Clocks**  
*Active cmd to internal read/write delay time,default is 0+1 clocks,max is 14+1 clocks.*
- uint8\_t **tRP\_Clocks**  
*Precharge cmd period-per bank,default is 0+1 clock,max is 14+1 clocks.*
- uint8\_t **tRPA\_Clocks**  
*Precharge cmd period-all bank,default is 0+1 clock,max is 14+1 clocks.*
- uint8\_t **tWR\_Clocks**  
*Write recovery time,default is 0+1 clock,max is 7+1 clocks.*
- uint8\_t **tMRD\_Clocks**  
*Mode register set cmd cycle,should set to max(tMRR,tMRW), default is 1+1 clock,max is 15+1 clocks.*
- uint8\_t **tRTP\_Clocks**

- `uint8_t tWTR_Clocks`  
*Internal read cmd to pre-charge cmd delay, default is 2+1 clock, max is 7+1 clocks.*
- `uint8_t tRRD_Clocks`  
*Internal write cmd to read cmd delay, default is 2+1 clock, max is 7+1 clocks.*
- `uint8_t tXP_Clocks`  
*active bankA to active bankB , Internal read cmd to pre-charge cmd, default is 0+1 clock, max is 6+1 clocks*
- `uint8_t tRSTtoCKE_Clocks`  
*exit power down to any cmd, default (1+1) clocks, max is (7 +1) clocks*
- `uint8_t tRTWSAME_Clocks`  
*idle time until first reset is assert, default is 14-2 clock, max is 0X3f-2 clocks, for LPDDR2 device default is 200us*
- `uint32_t tDAI_Clocks`  
*Maximum device auto initialization period for LPDDR2, not relavant to DDR3.*
- `uint8_t tRTWDIFF_Clocks`  
Read to write commands delay for same chip select, total delay is calculated according to:  $BL/2 + RTW\_SAME + (tCL-tCWL) + RALAT$
- `uint8_t tWTRDIFF_Clocks`  
Write to read commands delay for different chip select, total delay is calculated according to:  $BL/2 + WTR\_DIFF + (tCL-tCWL) + RALAT$
- `uint8_t tWTWDIFF_Clocks`  
Write to write commands delay for different chip select, total delay is calculated according to:  $BL/2 + WTW\_DIFF$
- `uint8_t tRTWDIFF_Clocks`  
Read to write commands delay for different chip select, total delay is calculated according to:  $BL/2 + RTW\_DIFF + (tCL - tCWL) + RALAT$
- `uint8_t tRTRDIFF_Clocks`  
*Read to read commands delay for different chip select.*
- `uint8_t tXPDLL_Clocks`  
*Exit precharge power down with DLL frozen to commands requiring DLL, not relavant to LPDDR2.*
- `uint16_t tDLLK_Clocks`  
*DLL locking time, not relavant to LPDDR2.*
- `uint8_t tXPR_Clocks`  
*CLKE High to a valid command, not relevant to LPDDR2.*
- `uint8_t tSDEtoRST_Clocks`  
*Time from SDE enable until DDR #reset is high, not relavant to LPDDR2.*
- `uint8_t tAOFPD_Clocks`  
*Asynchronous RTT turn-off delay, not relavant to LPDDR2.*
- `uint8_t tAONPD_Clocks`  
*Asynchronous RTT turn-on delay, not relavant to LPDDR2.*
- `uint8_t tODTIdleOff_Clocks`  
*ODT turn off latency, not relavant to LPDDR2.*

## 20.3.9.0.0.40 Field Documentation

### 20.3.9.0.0.40.1 `uint32_t mmdc_device_timing_t::tDAI_Clocks`

### 20.3.9.0.0.40.2 `uint8_t mmdc_device_timing_t::tRTRDIFF_Clocks`

total delay is calculated according to:  $BL/2 + RTR\_DIFF$

## Data Structure Documentation

- 20.3.9.0.0.40.3 uint8\_t mmdc\_device\_timing\_t::tXPDLL\_Clocks
- 20.3.9.0.0.40.4 uint16\_t mmdc\_device\_timing\_t::tDLLK\_Clocks
- 20.3.9.0.0.40.5 uint8\_t mmdc\_device\_timing\_t::tXPR\_Clocks
- 20.3.9.0.0.40.6 uint8\_t mmdc\_device\_timing\_t::tSDEtoRST\_Clocks
- 20.3.9.0.0.40.7 uint8\_t mmdc\_device\_timing\_t::tAOFPD\_Clocks
- 20.3.9.0.0.40.8 uint8\_t mmdc\_device\_timing\_t::tAONPD\_Clocks
- 20.3.9.0.0.40.9 uint8\_t mmdc\_device\_timing\_t::tODTIdleOff\_Clocks

### 20.3.10 struct mmdc\_auto\_refresh\_t

#### Data Fields

- uint16\_t refreshCnt
  - define refresh counter which is how many DDR clock cycles arrive will trigger auto refresh, only applied when choose refreshTrigSrc as kMMDC\_RefreshTrigDDRCycles
- uint16\_t refreshRate
  - refresh rate-means how much cmd will send once auto refresh being trigger
- mmdc\_refresh\_sel\_t refreshTrigSrc
  - select refresh trigger clock source

### 20.3.11 struct mmdc\_exaccess\_config\_t

#### Data Fields

- uint16\_t excMonitorID0
  - exclusive monitor ID 0
- uint16\_t excMonitorID1
  - exclusive monitor ID 1
- uint16\_t excMonitorID2
  - exclusive monitor ID 2
- uint16\_t excMonitorID3
  - exclusive monitor ID 3
- bool secErrLock
  - define if lock ARCR\_SEC\_ERR\_EN this bit can't update if locked
- bool secErrEn
  - This bit defines whether security read/write access violation result in SLV Error response or in OKAY response.
- bool excErrEn
  - This bit defines whether exclusive read/write access violation of AXI 6.2.4 rule result in SLV Error response or in OKAY response .

### 20.3.11.0.0.41 Field Documentation

#### 20.3.11.0.0.41.1 bool mmdc\_exaccess\_config\_t::excErrEn

Default value is 0x1 response is SLV Error

### 20.3.12 struct mmdc\_profiling\_config\_t

#### Data Fields

- `mmdc_profiling_action_t type`  
*profiling action*
- `bool overFlowCount`  
*profiling cycle counter over flag*
- `uint16_t axiIDMask`  
*profiling AXI ID mask*
- `uint16_t axiID`  
*profiling AXI ID*
- `uint32_t totalCount`  
*total cycle count-readonly*
- `uint32_t busyCount`  
*busy count-readonly*
- `uint32_t readCount`  
*total read count-readonly*
- `uint32_t writeCount`  
*total write count-readonly*
- `uint32_t readByteCount`  
*read byte count-readonly*
- `uint32_t writeByteCount`  
*total write byte count-readonly*

### 20.3.13 struct mmdc\_performance\_config\_t

#### Data Fields

- `bool enArbitration`  
*define if enable arbitration in MAARCR*
- `bool enReordering`  
*define if enable reordering in MAARCR*
- `bool enRCH`  
*define if enable real time channel in MAARCR*
- `uint32_t ratePageHit`  
*static score taken into account in case the pending access has a page hit in MAARCR*
- `uint32_t rateAccessHit`  
*static score taken into account in case the pending access  
is same as before in MAARCR*
- `uint32_t dynJump`

## Data Structure Documentation

- `uint32_t dynMax`  
*dynamic score give to any pending access in case it was not chosen in arbitration in MAARCR*
- `uint32_t guard`  
*dynamic score max value in MAARCR*
- `uint32_t cmdPredict`  
*use to prevent a starvation of access*
- `uint32_t cmdPredict`  
*define cmd prediction work mode in misc*

### 20.3.14 struct mmdc\_device\_config\_t

### 20.3.15 struct mmdc\_config\_t

## Data Fields

- `mmdc_device_type_t devType`  
*define device type*
- `uint32_t devSize`  
*define the size of the device*
- `mmdc_device_bank_num_t devBank`  
*define device total bank number*
- `mmdc_row_addr_width_t rowWidth`  
*define row width in MDCTL*
- `mmdc_col_addr_width_t colWidth`  
*define col width in MDCTL*
- `mmdc_burst_len_t burstLen`  
*define burst length MDCTL*
- `bool bankInterleave`  
*define indicate bank interleave on/off in misc*
- `bool secondDDRClock`  
*define gating the secondary DDR clock in misc*
- `bool enableOnlyCS0`  
*Only enable CS0.*
- `mmdc_odt_config_t * ODTConfig`  
*Pointer to on die termination config, NULL means disable, for LPDDR2, pass NULL.*
- `mmdc_zq_config_t * zqCalibration`  
*Pointer to device timing structure.*
- `mmdc_device_config_t * deviceConfig [2]`  
*Pointer to ZQ calibration config, NULL means do not need.*
- `mmdc_readDQS_calibration_config_t * readDQSCalibration [2]`  
*Pointer to device configuration CS0/CS1, NULL means do not need.*
- `mmdc_writeLeveling_calibration_config_t * wLevelingCalibration [2]`  
*Pointer to read DQS calibration config CS0/CS1, NULL means do not need, for LPDDR2, pass NULL.*
- `mmdc_read_calibration_config_t * readCalibration [2]`  
*Pointer to write leveling calibration config, NULL means do not need, for LPDDR2, pass NULL.*
- `mmdc_write_calibration_config_t * writeCalibration [2]`  
*Pointer to read calibration config CS0/CS1, NULL means do not need.*
- `mmdc_fine_tuning_config_t * tuning`  
*Pointer to write calibration config CS0/CS1, NULL means do not need.*
- `mmdc_auto_refresh_t * autoRefresh`

- `mmdc_power_config_t * powerConfig`  
*Pointer to auto refresh config structure, NULL means do not need.*

#### 20.3.15.0.0.42 Field Documentation

- 20.3.15.0.0.42.1 `mmdc_odt_config_t* mmdc_config_t::ODTConfig`
- 20.3.15.0.0.42.2 `mmdc_zq_config_t* mmdc_config_t::zqCalibration`
- 20.3.15.0.0.42.3 `mmdc_device_config_t* mmdc_config_t::deviceConfig[2]`
- 20.3.15.0.0.42.4 `mmdc_readDQS_calibration_config_t* mmdc_config_t::readDQSCalibration[2]`
- 20.3.15.0.0.42.5 `mmdc_writeLeveling_calibration_config_t* mmdc_config_t::wLevelingCalibration[2]`
- 20.3.15.0.0.42.6 `mmdc_read_calibration_config_t* mmdc_config_t::readCalibration[2]`
- 20.3.15.0.0.42.7 `mmdc_write_calibration_config_t* mmdc_config_t::writeCalibration[2]`
- 20.3.15.0.0.42.8 `mmdc_fine_tuning_config_t* mmdc_config_t::tuning`
- 20.3.15.0.0.42.9 `mmdc_auto_refresh_t* mmdc_config_t::autoRefresh`
- 20.3.15.0.0.42.10 `mmdc_power_config_t* mmdc_config_t::powerConfig`

### 20.4 Macro Definition Documentation

#### 20.4.1 #define MMDC\_READ\_DQS\_FINE\_TUNING\_MASK (0x77777777U)

define the read DQS fine tuning mask value

#### 20.4.2 #define MMDC\_WRITE\_DQS\_FINE\_TUNING\_MASK (0xF3333333U)

define the write DQS fine tuning mask value

#### 20.4.3 #define MMDC\_PRE\_DEFINE\_VALUE\_DEFAULT (0xCCU)

define the calibration predefined value

#### 20.4.4 #define MMDC\_MEASUREUNIT\_ERR\_FREQ (100000000U)

according to ERR005778 description

## Enumeration Type Documentation

### 20.5 Typedef Documentation

20.5.1 `typedef void(* MMDC_SwitchFrequency)(MMDC_Type *, void *, void *, uint32_t)`

### 20.6 Enumeration Type Documentation

#### 20.6.1 `enum _mmdc_status`

Enumerator

*kStatus\_MMDC\_ErrorDGCalibration* MMDC error DG calibration.

*kStatus\_MMDC\_ErrorReadCalibration* MMDC error read calibration.

*kStatus\_MMDC\_ErrorWriteCalibration* MMDC error write calibration.

*kStatus\_MMDC\_ErrorWriteLevelingCalibration* MMDC error write leveling calibration.

*kStatus\_MMDC\_WaitFlagTimeout* MMDC wait flag timeout.

#### 20.6.2 `enum mmdc_device_type_t`

Enumerator

*kMMDC\_LPDDR2\_S4* LPDDR2-S4.

*kMMDC\_LPDDR2\_S2* LPDDR2-S2.

*kMMDC\_DDR3* DDR3 device.

#### 20.6.3 `enum mmdc_device_bank_num_t`

Enumerator

*kMMDC\_Bank8* bank number 8

*kMMDC\_Bank4* bank number 4

#### 20.6.4 `enum mmdc_row_addr_width_t`

Enumerator

*kMMDC\_Row11Bits* row addr 11 bits

*kMMDC\_Row12Bits* row addr 12 bits

*kMMDC\_Row13Bits* row addr 13 bits

*kMMDC\_Row14Bits* row addr 14 bits

*kMMDC\_Row15Bits* row addr 15 bits

*kMMDC\_Row16Bits* row addr 16 bits

## 20.6.5 enum mmdc\_col\_addr\_width\_t

Enumerator

- kMMDC\_Col9Bits* col addr 9 bits
- kMMDC\_Col10Bits* col addr 10 bits
- kMMDC\_Col11Bits* col addr 11 bits
- kMMDC\_Col8Bits* col addr 8 bits
- kMMDC\_Col12Bits* col addr 12 bits

## 20.6.6 enum mmdc\_burst\_len\_t

Enumerator

- kMMDC\_BurstLen4* brust len 4
- kMMDC\_BurstLen8* burst len 8
- kMMDC\_BurstLen16* reserved

## 20.6.7 enum mmdc\_cmd\_type\_t

Enumerator

- kMMDC\_NormalOperation* normal operation cmd
- kMMDC\_AutoRefresh* auto refresh cmd
- kMMDC\_WriteModeRegister* load mode register for DDR2/DDR3,MRW for LPDDR2
- kMMDC\_ZQCalibration* ZQ calibration cmd.
- kMMDC\_PreChargeAll* Precharge all cmd.
- kMMDC\_ReadModeRegister* mode register read cmd

## 20.6.8 enum mmdc\_zq\_calmode\_t

Enumerator

- kMMDC\_ZQCaltoIOHW* ZQ calibration to IO pads only through HW.
- kMMDC\_ZQCaltoIODeviceLong* ZQ calibration to IO pads together with ZQ long cmd to device.
- kMMDC\_ZQCaltoDeviceOnly* ZQ calibration to device with long/short cmd.
- kMMDC\_ZQCaltoIODeviceLongShort* ZQ calibration to IO pads together with ZQ calibration cmd long/short to device.
- kMMDC\_ZQFinetuning* HW ZQ res offset fine tuning.
- kMMDC\_DisZQFinetuning* disable HW ZQ res offset

## Enumeration Type Documentation

### 20.6.9 enum mmdc\_zq\_calfreq\_t

Enumerator

*kMMDC\_ZQCalFreq1ms* ZQ calibration is performed every 1ms.  
*kMMDC\_ZQCalFreq2ms* ZQ calibration is performed every 2ms.  
*kMMDC\_ZQCalFreq4ms* ZQ calibration is performed every 4ms.  
*kMMDC\_ZQCalFreq1s* ZQ calibration is performed every 1s.  
*kMMDC\_ZQCalFreq16s* ZQ calibration is performed every 16s.  
*kMMDC\_ZQCalFreq32s* ZQ calibration is performed every 32s.

### 20.6.10 enum mmdc\_refresh\_sel\_t

Enumerator

*kMMDC\_RefreshTrigBy64K* refresh trigger frequency 64K  
*kMMDC\_RefreshTrigBy32K* refresh trigger frequency 32K  
*kMMDC\_RefreshTrigDDRCycles* refresh trigger every amount of cycles that are configured in RE-F\_CNT field  
*kMMDC\_RefreshTrigNone* auto refresh disable

### 20.6.11 enum mmdc\_profiling\_action\_t

Enumerator

*kMMDC\_EnProfilingWithID* enable profiling with special ID  
*kMMDC\_FreezeProfiling* freeze profiling  
*kMMDC\_CheckOverFlow* check the counter overflow

### 20.6.12 enum mmdc\_calibration\_type\_t

Enumerator

*kMMDC\_CalWithPreSetValue* calibration with preset value  
*kMMDC\_CalWithMPR* HW calibration with the MPR.  
*kMMDC\_CalWithPreDefine* calibration with Pre-defined value

### 20.6.13 enum mmdc\_calibaration\_waitcycles\_t

Enumerator

*kMMDC\_Wait16DDRCycles* wait 16 DDR cycles before comparing sample data

*kMMDC\_Wait32DDRCycles* wait 32 DDR cycles before comparing sample data

## 20.6.14 enum mmdc\_fine\_tuning\_dutycycle\_t

Enumerator

*kMMDC\_DutyHighPercent48\_5* 51.5% low 48.5% high

*kMMDC\_DutyHighPercent50* 50% duty cycle

*kMMDC\_DutyHighPercent51\_5* 48.5% low 51.5% high

## 20.6.15 enum mmdc\_termination\_config\_t

Enumerator

*kMMDC\_RttNomDisabled* Rtt\_Nom Disabled.

*kMMDC\_RttNom120ohm* Rtt\_Nom 120 Ohm.

*kMMDC\_RttNom60ohm* Rtt\_Nom 60 Ohm.

*kMMDC\_RttNom40ohm* Rtt\_Nom 40 Ohm.

*kMMDC\_RttNom30ohm* Rtt\_Nom 30 Ohm.

*kMMDC\_RttNom24ohm* Rtt\_Nom 24 Ohm.

*kMMDC\_RttNom20ohm* Rtt\_Nom 20 Ohm.

*kMMDC\_RttNom17ohm* Rtt\_Nom 17 Ohm.

## 20.6.16 enum \_mmdc\_lpddr2\_derate

Enumerator

*kMMDC\_NoUpdateDerate* no derate

*kMMDC\_UpdateRefreshRate* refresh rate derate

*kMMDC\_DerateTiming* derating relate timing

## 20.6.17 enum \_mmdc\_exaccess\_type

Enumerator

*kMMDC\_ExMonitorID0* config the exclusive access ID0

*kMMDC\_ExMonitorID1* config the exclusive access ID1

*kMMDC\_ExMonitorID2* config the exclusive access ID2

*kMMDC\_ExMonitorID3* config the exclusive access ID3

*kMMDC\_ExAccessResponse* config the exclusive access reponse

## Function Documentation

### 20.7 Function Documentation

20.7.1 **void MMDC\_GetDefaultConfig ( mmdc\_config\_t \* *config* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>mmdc</i>	config collection pointer config->bankInterleave = true; config->secondDDR-Clock = true; config->enableOnlyCS0 = true; config->devType = kMMDC_-DDR3; config->devSize = 0x40000000U; config->devBank = kMMDC_Bank8; config->rowWidth = kMMDC_Row16Bits; config->colWidth = kMMDC_Col10-Bits; config->burstLen = kMMDC_BurstLen8; config->ODTConfig = NULL; config->timing = NULL; config->zqCalibration = NULL; config->deviceConfig[0] = NULL; config->deviceConfig[1] = NULL; config->readDQSCalibration[0] = N-ULL; config->readDQSCalibration[1] = NULL; config->wLevelingCalibration[0] = NULL; config->wLevelingCalibration[1] = NULL; config->readCalibration[0] = NULL; config->readCalibration[1] = NULL; config->writeCalibration[0] = NUL-L; config->writeCalibration[1] = NULL; config->tuning = NULL; config->auto-Refresh = NULL; config->powerConfig = NULL;

## 20.7.2 **status\_t MMDC\_Init ( MMDC\_Type \* *base*, mmdc\_config\_t \* *config* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>mmdc</i>	config collection pointer

Return values

<i>kStatus_Success</i>	Initialization succeed
<i>kStatus_MMDC_ErrorD-GCalibration</i>	Error happened during hardware DQS gate calibration
<i>kStatus_MMDC_Error-ReadCalibration</i>	Error happened during hardware read calibration
<i>kStatus_MMDC_Error-WriteCalibration</i>	Error happened during hardware write calibration
<i>kStatus_MMDC_Error-WriteLevelingCalibration</i>	Error happened during hardware write leveling calibration

## 20.7.3 **void MMDC\_Deinit ( MMDC\_Type \* *base* )**

## Function Documentation

Parameters

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

**20.7.4 void MMDC\_HandleCommand ( MMDC\_Type \* *base*, mmdc\_cmd\_config\_t \* *config* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>cmd</i>	configuration collection

**20.7.5 status\_t MMDC\_GetReadData ( MMDC\_Type \* *base*, uint32\_t \* *data* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>the</i>	pointer which used to store read data

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_Wait-FlagTimeout</i>	Read data flag wait timeout

**20.7.6 void MMDC\_EnhancePerformance ( MMDC\_Type \* *base*, const mmdc\_performance\_config\_t \* *config* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>performance</i>	configuration collection

**20.7.7 void MMDC\_EnableAutoRefresh ( MMDC\_Type \* *base*, mmdc\_auto\_refresh\_t \* *config* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>mmdc</i>	auto refresh configuration collection

#### 20.7.8 static void MMDC\_DisableAutoRefresh ( **MMDC\_Type** \* *base* ) [inline], [static]

Parameters

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

#### 20.7.9 void MMDC\_EnablePowerSaving ( **MMDC\_Type** \* *base*, **mmdc\_power\_config\_t** \* *config* )

Parameters

<i>base</i>	MMDC peripheral base address
<i>mmdc</i>	device configuration collection, pointer for the configuration.

#### 20.7.10 static void MMDC\_DisablePowerSaving ( **MMDC\_Type** \* *base* ) [inline], [static]

MMDC disable automatic power saving.

Parameters

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

#### 20.7.11 void MMDC\_Profiling ( **MMDC\_Type** \* *base*, **mmdc\_profiling\_config\_t** \* *config* )

## Function Documentation

Parameters

<i>base</i>	MMDC peripheral base address
<i>mmdc</i>	profiling status and control

### 20.7.12 **status\_t MMDC\_LPDDR2UpdateDerate ( MMDC\_Type \* *base*, mmdc\_auto\_refresh\_t \* *config*, uint32\_t *type* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>auto</i>	refresh configuration collection,can set to NULL,when do not change refresh rate
<i>derating</i>	type

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_WaitFlagTimeout</i>	LPDDR2 AC timing/refresh derate wait flag timeout

### 20.7.13 **status\_t MMDC\_MonitorLPDDR2OperationTemp ( MMDC\_Type \* *base*, uint32\_t \* *mr4* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>MR4</i>	pointer,use to store the mode register4 value

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_WaitFlagTimeout</i>	Get MR4 data flag timeout

### 20.7.14 **status\_t MMDC\_ReadDQS\_GatingCalibration ( MMDC\_Type \* *base*, mmdc\_readDQS\_calibration\_config\_t \* *config* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>config</i>	calibration configuration collection

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_ErrorD-GCalibration</i>	Read DQS data gate hardware calibration error

### 20.7.15 status\_t MMDC\_WriteLevelingCalibration ( **MMDC\_Type** \* *base*, **mmdc\_writeLeveling\_calibration\_config\_t** \* *config* )

Parameters

<i>base</i>	MMDC peripheral base address
<i>config</i>	calibration configuration collection

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_Error-WriteLevelingCalibration</i>	write leveling hardware calibration error

### 20.7.16 status\_t MMDC\_WriteCalibration ( **MMDC\_Type** \* *base*, **mmdc\_write\_calibration\_config\_t** \* *config* )

Parameters

<i>base</i>	MMDC peripheral base address
<i>config</i>	calibration configuration collection

Return values

## Function Documentation

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_Error-WriteCalibration</i>	write hardware calibration error

**20.7.17 `status_t MMDC_ReadCalibration ( MMDC_Type * base, mmdc_read_calibration_config_t * config )`**

Parameters

<i>base</i>	MMDC peripheral base address
<i>config</i>	calibration configuration collection

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_Error-ReadCalibration</i>	read hardware calibration error

**20.7.18 `void MMDC_DoFineTuning ( MMDC_Type * base, mmdc_device_type_t devType, mmdc_fine_tuning_config_t * config )`**

Parameters

<i>base</i>	MMDC peripheral base address
<i>devType</i>	MMDC device type
<i>config</i>	fine tuning configuration collection

**20.7.19 `void MMDC_SetTiming ( MMDC_Type * base, mmdc_device_type_t devType, mmdc_device_timing_t * timing )`**

Parameters

<i>MMDC</i>	peripheral base address
<i>timing</i>	pointer to timing structure

20.7.20 **void MMDC\_DeviceInit ( MMDC\_Type \* *base*, mmdc\_device\_type\_t *devType*, uint8\_t *targetCS*, mmdc\_device\_config\_t \* *devConfig* )**

## Function Documentation

Parameters

<i>MMDC</i>	base address
<i>device</i>	basic config info pointer

**20.7.21 status\_t MMDC\_EnterConfigurationMode ( *MMDC\_Type* \* *base*, *bool enable* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>enable</i>	enter/exit flag

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_Wait-FlagTimeout</i>	Enter configuration mode time out

**20.7.22 void MMDC\_DoZQCalibration ( *MMDC\_Type* \* *base*, *mmdc\_device\_type\_t devType*, *mmdc\_zq\_config\_t* \* *zqCal* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>devType</i>	device type
<i>zqCal</i>	info pointer

**20.7.23 status\_t MMDC\_EnableLowPowerMode ( *MMDC\_Type* \* *base*, *bool enable* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_WaitFlagTimeout</i>	Enter low power mode timeout

### 20.7.24 **status\_t MMDC\_EnableDVFSMode ( MMDC\_Type \* *base*, bool *enable* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_Success</i>	Read data succeed
<i>kStatus_MMDC_WaitFlagTimeout</i>	Enter DVFS mode timeout

### 20.7.25 **void MMDC\_Reset ( MMDC\_Type \* *base* )**

Parameters

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

### 20.7.26 **static void MMDC\_SwitchDeviceFrequency ( MMDC\_Type \* *base*, void \* *ccm*, void \* *iomux*, uint32\_t *param*, uint32\_t *codeAddr* ) [inline], [static]**

Parameters

<i>MMDC</i>	base address
<i>CCM</i>	base address
<i>iomux</i>	base address
<i>target</i>	frequency value for LPDDR2 and parameter address for DDR3
<i>assembly</i>	switch frequency code address

## Function Documentation

20.7.27 **static void MMDC\_EnableSBS ( MMDC\_Type \* *base*, bool *enable* )**  
[**inline**], [**static**]

Parameters

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

**20.7.28 static void MMDC\_TriggerSBS ( MMDC\_Type \* *base* ) [inline], [static]**

Parameters

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

**20.7.29 static uint32\_t MMDC\_GetAXIAddrBySBS ( MMDC\_Type \* *base* ) [inline], [static]**

Parameters

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

**20.7.30 static uint32\_t MMDC\_GetAXIAttributeBySBS ( MMDC\_Type \* *base* ) [inline], [static]**

Parameters

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

**20.7.31 static void MMDC\_EnableProfiling ( MMDC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

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

## Function Documentation

<i>enable</i>	or disable flag
---------------	-----------------

**20.7.32 static void MMDC\_ResumeProfiling ( MMDC\_Type \* *base* ) [inline], [static]**

Parameters

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

**20.7.33 static void MMDC\_ResetProfiling ( MMDC\_Type \* *base* ) [inline], [static]**

Parameters

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

**20.7.34 void MMDC\_ExclusiveAccess ( MMDC\_Type \* *base*, mmdc\_exaccess\_config\_t \* *config*, uint32\_t *type* )**

Parameters

<i>base</i>	MMDC peripheral base address
<i>exclusive</i>	access config collection
<i>exclusive</i>	access config type

# Chapter 21

## PMU: Power Management Unit

### 21.1 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_gate` {  
    `kPMU_PowerGateDisplay` = PMU\_LOWPWR\_CTRL\_MIX\_PWRGATE\_MASK,  
    `kPMU_PowerGateDisplayLogic` = PMU\_LOWPWR\_CTRL\_DISPLAY\_PWRGATE\_MASK,  
    `kPMU_PowerGateL2` = PMU\_LOWPWR\_CTRL\_L2\_PWRGATE\_MASK,  
    `kPMU_PowerGateL1` = PMU\_LOWPWR\_CTRL\_L1\_PWRGATE\_MASK,  
    `kPMU_PowerGateRefTopIBias` = PMU\_LOWPWR\_CTRL\_REFTOP\_IBIAS\_OFF\_MASK }

## Overview

- *Mask values of power gate.*  
enum pmu\_power\_bandgap\_t {  
    kPMU\_NormalPowerBandgap = 0U,  
    kPMU\_LowPowerBandgap = 1U }  
*Bandgap select.*

## 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\\_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.*

## Power Gate Controller & other

- static void [PMU\\_GatePower](#) (PMU\_Type \*base, uint32\_t gates)  
*Gate the power to modules.*
- static void [PMU\\_UngatePower](#) (PMU\_Type \*base, uint32\_t gates)  
*Ungate the power to modules.*
- static void [PMU\\_EnableLowPowerBandgap](#) (PMU\_Type \*base, bool enable)  
*Enable the low power bandgap.*

## 21.2 Macro Definition Documentation

### 21.2.1 #define FSL\_PMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

## Enumeration Type Documentation

### 21.3 Enumeration Type Documentation

#### 21.3.1 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.

#### 21.3.2 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.

#### 21.3.3 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.

#### 21.3.4 enum pmu\_core\_reg\_voltage\_ramp\_rate\_t

Enumerator

***kPMU\_CoreRegVoltageRampRateFast*** Fast.

***kPMU\_CoreRegVoltageRampRateMediumFast*** Medium Fast.

***kPMU\_CoreRegVoltageRampRateMediumSlow*** Medium Slow.

***kPMU\_CoreRegVoltageRampRateSlow*** Slow.

### 21.3.5 enum \_pmu\_power\_gate

Enumerator

*kPMU\_PowerGateDisplay* Display power gate control.  
*kPMU\_PowerGateDisplayLogic* Display logic power gate control.  
*kPMU\_PowerGateL2* L2 power gate control.  
*kPMU\_PowerGateL1* L1 power gate control.  
*kPMU\_PowerGateRefTopIBias* Low power reftop ibias disable.

### 21.3.6 enum pmu\_power\_bandgap\_t

Enumerator

*kPMU\_NormalPowerBandgap* Normal power bandgap.  
*kPMU\_LowPowerBandgap* Low power bandgap.

## 21.4 Function Documentation

### 21.4.1 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> .

### 21.4.2 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.

## Function Documentation

### 21.4.3 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.

### 21.4.4 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.

Parameters

<i>base</i>	PMU peripheral base address.
<i>value</i>	Setting value for the brownout offset. The available range is in 3-bit.

### 21.4.5 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.

### 21.4.6 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.

#### 21.4.7 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.

#### 21.4.8 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.

#### 21.4.9 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.

## Function Documentation

**21.4.10 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.

#### 21.4.11 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.

Parameters

<i>base</i>	PMU peripheral base address.
<i>value</i>	Setting value for the brownout offset. The available range is in 3-bit.

#### 21.4.12 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.

#### 21.4.13 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.

#### 21.4.14 static void PMU\_3P0EnableOutput ( 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.

### 21.4.15 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.

### 21.4.16 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.

### 21.4.17 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.

**21.4.18 static void PMU\_2P1EnablePullDown ( PMU\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

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

**21.4.19 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.

**21.4.20 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.

**21.4.21 static void PMU\_2P5EnableOutput ( 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.

### 21.4.22 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.

### 21.4.23 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> .

### 21.4.24 static void PMU\_CoreSetSOCDomainVoltage ( PMU\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Define the target voltage for the SOC power domain. Single-bit increments reflect 25mV core voltage steps. Some steps may not be relevant because of input supply limitations or load operation.

- 0x00 : Power gated off.
- 0x01 : Target core voltage = 0.725V
- 0x02 : Target core voltage = 0.750V
- ...
- 0x10 : Target core voltage = 1.100V
- ...
- 0x1e : Target core voltage = 1.450V

- 0x1F : Power FET switched full on. No regulation. NOTE: This register is capable of programming an over-voltage condition on the device. Consult the datasheet Operating Ranges table for the allowed voltages.

Parameters

<i>base</i>	PMU peripheral base address.
<i>value</i>	Setting value for target voltage. 5-bit available

#### 21.4.25 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

#### 21.4.26 static void PMU\_GatePower ( PMU\_Type \* *base*, uint32\_t *gates* ) [inline], [static]

Parameters

<i>base</i>	PMU peripheral base address.
<i>gates</i>	Mask value for the module to be gated. See to <a href="#">_pmu_power_gate</a> .

## Function Documentation

21.4.27 **static void PMU\_UngatePower ( PMU\_Type \* *base*, uint32\_t *gates* )**  
[**inline**], [**static**]

Parameters

<i>base</i>	PMU peripheral base address.
<i>gates</i>	Mask value for the module to be gated. See to <a href="#">_pmu_power_gate</a> .

#### 21.4.28 static void PMU\_EnableLowPowerBandgap ( PMU\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	PMU peripheral base address.
<i>enable</i>	Enable the low power bandgap or use the normal power bandgap. @

## Function Documentation

# Chapter 22

## PXP: Pixel Pipeline

### 22.1 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 the configure input buffer, output buffer, and other setting 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.

### 22.2 Typical use case

#### 22.2.1 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.

```
/* Initialize the PXP first. */
PXP_Init(PXP);

/* PS configure. */
const ppxp_ps_buffer_config_t psBufferConfig =
{
    .pixelFormat = kPXP_PsPixelFormatRGB888,
    .swapByte = false,
    .bufferAddr = psBufferAddr,
    .bufferAddrU = 0U,
    .bufferAddrV = 0U,
    .pitchBytes = PS_BUF_PITCH,
};

const ppxp_output_buffer_config_t outputBufferConfig =
{
    .pixelFormat = kPXP_OutputPixelFormatRGB888,
    .interlacedMode = kPXP_OutputProgressive,
    .buffer0Addr = outputBufferAddr,
    .buffer1Addr = 0U,
    .pitchBytes = OUT_BUF_PITCH,
    .width = IMG_WIDTH,
    .height = IMG_HEIGHT,
};

PXP_SetProcessSurfaceBackGroundColor(PXP, 0U);

PXP_SetProcessSurfaceBufferConfig(PXP, &psBufferConfig);
```

## Typical use case

```
PXP_SetProcessSurfacePosition(PXP, PS_ULC_X, PS_ULC_Y, PS_LRC_X, PS_LRC_Y);

/* Disable AS if not used. */
PXP_SetAlphaSurfacePosition(PXP, 0xFFFFU, 0xFFFFU, 0U, 0U);

/* Output config. */
PXP_SetOutputBufferConfig(PXP, &outputBufferConfig);

/* Disable CSC1, it is enabled by default. */
PXP_EnableCsc1(PXP, false);

/* Set FLIP configuration */
PXP_SetRotateConfig(PXP, kPXP_RotateOutputBuffer,
    kPXP_Rotate0, kPXP_FlipVertical);

/* Start processing. */
PXP_Start(PXP);

/* Wait for process complete. Interrupt method could be used alternatively. */
while (!(kPXP_CompleteFlag & PXP_GetStatusFlags(PXP)))
{
}
```

### 22.2.2 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.

```
/* Initialize the PXP first. */
PXP_Init(PXP);

/* The commands are organized following PXP memory map. */

static uint32_t ppxCommands[] =
{
    PXP_CTRL_VFLIP_MASK | PXP_CTRL_ENABLE_MASK, /* CTRL */
    0x00000000, /* STAT, don't care */
    PXP_PS_CTRL_FORMAT(kPXP_OutputPixelFormatRGB888), /* OUT_CTRL */
    outputBufferAddr, /* OUT_BUF */
    0x00000000, /* OUT_BUF2 */
    OUT_PITCH, /* OUT_PITCH */
    PXP_OUT_LRC_Y(IMG_HEIGHT - 1U) | PXP_OUT_LRC_X(IMG_WIDTH - 1U), /* OUT_LRC */
    PXP_OUT_PS_ULC_Y(PS_ULC_Y) | PXP_OUT_PS_ULC_X(PS_ULC_X), /* OUT_PS_ULC */
    PXP_OUT_PS_LRC_Y(PS_LRC_Y) | PXP_OUT_PS_LRC_X(PS_LRC_X), /* OUT_PS_LRC */

    /* Disable AS. */
    0x3FFF3FFF, /* OUT_AS_ULC */
    0x00000000, /* OUT_AS_LRC */
    0x00000004, /* PS_CTRL */
    bufferAddr, /* PS_BUF */
    0x00000000, /* PS_UBUF */
    0x00000000, /* PS_VBUF */
    PS_PITCH, /* PS_PITCH */
    0x00000000, /* PS_BACHGROUND */
    0x10001000, /* PS_SCALE */
    0x00000000, /* PS_OFFSET */
    0x00FFFFFF, /* PS_CLRKEYLOW */
    0x00000000, /* PS_CLRKEYHIGH */
    0x00000000, /* AS_CTRL */
    0x00000000, /* AS_BUF */
    0x00000000, /* AS_PITCH */
    0x00FFFFFF, /* AS_CLRKEYLOW */
    0x00000000, /* AS_CLRKEYHIGH */
```

```

PXP_CSC1_COEF0_BYPASS_MASK, /* CSC1_COEF0, don't care. */
0x00000000, /* CSC1_COEF1, don't care. */
0x00000000, /* CSC1_COEF2, don't care. */
PXP_CSC2_CTRL_BYPASS_MASK, /* CSC2_CTRL */
0x00000000, /* CSC2_COEF0, don't care. */
0x00000000, /* CSC2_COEF1, don't care. */
0x00000000, /* CSC2_COEF2, don't care. */
0x00000000, /* CSC2_COEF3, don't care. */
0x00000000, /* CSC2_COEF4, don't care. */
0x00000000, /* CSC2_COEF5, don't care. */
PXP_LUT_CTRL_BYPASS_MASK, /* LUT_CTRL */
0x00000000, /* LUT_ADDR */
0x00000000, /* LUT_DATA */
0x00000000, /* LUT_EXTMEM */
0x00000000, /* CFA */
0x00000020, /* HIST_CTRL */
0x00000F00, /* HIST2_PARAM */
0x0F0A0500, /* HIST4_PARAM */
0x06040200, /* HIST8_PARAM0 */
0x0F0D0B09, /* HIST8_PARAM1 */
0x03020100, /* HIST16_PARAM0 */
0x07060504, /* HIST16_PARAM1 */
0x0B0A0908, /* HIST16_PARAM2 */
0x0F0E0D0C, /* HIST16_PARAM3 */
0x00000000, /* POWER */
0x00000000, /* NEXT, don't care */
};

while (PXP_IsNextCommandPending(PXP))
{
}
PXP_SetNextCommand(PXP, ppxCommands);

/* Wait for process complete. Interrupt method could be used alternatively. */
while (!(kPXP_CompleteFlag & PXP_GetStatusFlags(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 alphas surface buffer configuration.* [More...](#)
- struct [pxp\\_as\\_blend\\_config\\_t](#)  
*PXP alpha surface blending configuration.* [More...](#)
- struct [pxp\\_csc2\\_config\\_t](#)  
*PXP CSC2 configuration.* [More...](#)
- struct [pxp\\_lut\\_config\\_t](#)  
*PXP LUT configuration.* [More...](#)
- struct [pxp\\_dither\\_final\\_lut\\_data\\_t](#)  
*PXP dither final LUT data.* [More...](#)
- struct [pxp\\_dither\\_config\\_t](#)  
*PXP dither configuration.* [More...](#)

## Typical use case

### Enumerations

- enum `_pxp_interrupt_enable` {  
    `kPXP_CommandLoadInterruptEnable` = `PXP_CTRL_NEXT_IRQ_ENABLE_MASK`,  
    `kPXP_CompleteInterruptEnable` = `PXP_CTRL_IRQ_ENABLE_MASK`,  
    `kPXP_LutDmaLoadInterruptEnable` = `PXP_CTRL_LUT_DMA_IRQ_ENABLE_MASK` }  
    *PXP interrupts to enable.*
- enum `_pxp_flags` {  
    `kPXP_CommandLoadFlag` = `PXP_STAT_NEXT_IRQ_MASK`,  
    `kPXP_CompleteFlag` = `PXP_STAT_IRQ0_MASK`,  
    `kPXP_LutDmaLoadFlag` = `PXP_STAT_LUT_DMA_LOAD_DONE_IRQ_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_OutputPixelFormatARGB888 = 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 ppxp_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 ppxp_block_size_t {  
    kPXP_BlockSize8 = 0U,  
    kPXP_BlockSize16 }  
    PXP process block size.  
• enum ppxp_csc1_mode_t {  
    kPXP_Csc1YUV2RGB = 0U,  
    kPXP_Csc1YCbCr2RGB }  
        PXP CSC1 mode.  
• enum ppxp_csc2_mode_t {  
    kPXP_Csc2YUV2RGB = 0U,  
    kPXP_Csc2YCbCr2RGB,  
    kPXP_Csc2RGB2YUV,  
    kPXP_Csc2RGB2YCbCr }  
        PXP CSC2 mode.  
• enum ppxp_lut_lookup_mode_t {  
    kPXP_LutCacheRGB565 = 0U,  
    kPXP_LutDirectY8,  
    kPXP_LutDirectRGB444,  
    kPXP_LutDirectRGB454 }  
        PXP LUT lookup mode.  
• enum ppxp_lut_out_mode_t {  
    kPXP_LutOutY8 = 1U,  
    kPXP_LutOutRGBW4444CFA,  
    kPXP_LutOutRGB888 }  
        PXP LUT output mode.  
• enum ppxp_lut_8k_bank_t {  
    kPXP_Lut8kBank0 = 0U,
```

- `kPXP_Lut8kBank1 }`  
*PXP LUT 8K bank index used when lookup mode is `kPXP_LutDirectRGB444`.*
- 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_EnableContinousRun` (PXP\_Type \*base, bool enable)  
*Enable or disable continous 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)

## Typical use case

- static void **PXP\_ClearStatusFlags** (PXP\_Type \*base, uint32\_t statusMask)  
*Gets PXP status flags.*  
*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.*

## 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 output 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, ppx\_rotate\_position\_t position, ppx\_rotate\_degree\_t degree, ppx\_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_SetCsc2Config` (PXP\_Type \*base, const ppx\_csc2\_config\_t \*config)  
*Set the CSC2 configuration.*
- static void `PXP_EnableCsc2` (PXP\_Type \*base, bool enable)  
*Enable or disable the CSC2.*
- void `PXP_SetCsc1Mode` (PXP\_Type \*base, ppx\_csc1\_mode\_t mode)  
*Set the CSC1 mode.*
- static void `PXP_EnableCsc1` (PXP\_Type \*base, bool enable)  
*Enable or disable the CSC1.*

## LUT operations

- void `PXP_SetLutConfig` (PXP\_Type \*base, const ppx\_lut\_config\_t \*config)  
*Set the LUT configuration.*
- status\_t `PXP_LoadLutTable` (PXP\_Type \*base, ppx\_lut\_lookup\_mode\_t lookupMode, uint32\_t bytesNum, uint32\_t memAddr, uint16\_t lutStartAddr)  
*Set the look up table to PXP.*
- static void `PXP_EnableLut` (PXP\_Type \*base, bool enable)  
*Enable or disable the LUT.*
- static void `PXP_Select8kLutBank` (PXP\_Type \*base, ppx\_lut\_8k\_bank\_t bank)  
*Select the 8kB LUT bank in DIRECT\_RGB444 mode.*

## 22.3 Data Structure Documentation

### 22.3.1 struct ppx\_output\_buffer\_config\_t

#### Data Fields

- ppx\_output\_pixel\_format\_t pixelFormat  
*Output buffer pixel format.*
- ppx\_interlaced\_output\_mode\_t interlacedMode

## Data Structure Documentation

- *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.*

### 22.3.1.0.0.43 Field Documentation

22.3.1.0.0.43.1 `pxp_output_pixel_format_t ppx_output_buffer_config_t::pixelFormat`

22.3.1.0.0.43.2 `pxp_interlaced_output_mode_t ppx_output_buffer_config_t::interlacedMode`

22.3.1.0.0.43.3 `uint32_t ppx_output_buffer_config_t::buffer0Addr`

22.3.1.0.0.43.4 `uint32_t ppx_output_buffer_config_t::buffer1Addr`

22.3.1.0.0.43.5 `uint16_t ppx_output_buffer_config_t::pitchBytes`

22.3.1.0.0.43.6 `uint16_t ppx_output_buffer_config_t::width`

22.3.1.0.0.43.7 `uint16_t ppx_output_buffer_config_t::height`

### 22.3.2 `struct ppx_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.*

**22.3.2.0.0.44 Field Documentation****22.3.2.0.0.44.1 pxp\_ps\_pixel\_format\_t pxp\_ps\_buffer\_config\_t::pixelFormat****22.3.2.0.0.44.2 bool pxp\_ps\_buffer\_config\_t::swapByte****22.3.2.0.0.44.3 uint32\_t pxp\_ps\_buffer\_config\_t::bufferAddr****22.3.2.0.0.44.4 uint32\_t pxp\_ps\_buffer\_config\_t::bufferAddrU****22.3.2.0.0.44.5 uint32\_t pxp\_ps\_buffer\_config\_t::bufferAddrV****22.3.2.0.0.44.6 uint16\_t pxp\_ps\_buffer\_config\_t::pitchBytes****22.3.3 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.*

**22.3.3.0.0.45 Field Documentation****22.3.3.0.0.45.1 pxp\_as\_pixel\_format\_t pxp\_as\_buffer\_config\_t::pixelFormat****22.3.3.0.0.45.2 uint32\_t pxp\_as\_buffer\_config\_t::bufferAddr****22.3.3.0.0.45.3 uint16\_t pxp\_as\_buffer\_config\_t::pitchBytes****22.3.4 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

### 22.3.4.0.0.46 Field Documentation

22.3.4.0.0.46.1 `uint8_t ppx_as_blend_config_t::alpha`

22.3.4.0.0.46.2 `bool ppx_as_blend_config_t::invertAlpha`

22.3.4.0.0.46.3 `ppx_alpha_mode_t ppx_as_blend_config_t::alphaMode`

22.3.4.0.0.46.4 `ppx_rop_mode_t ppx_as_blend_config_t::ropMode`

### 22.3.5 `struct ppx_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

- `ppx_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.*
- `uint16_t D1`  
*D1.*
- `uint16_t D2`  
*D2.*
- `uint16_t D3`  
*D3.*

### 22.3.5.0.0.47 Field Documentation

22.3.5.0.0.47.1 `pxp_csc2_mode_t pxp_csc2_config_t::mode`

22.3.5.0.0.47.2 `float pxp_csc2_config_t::A1`

22.3.5.0.0.47.3 `float pxp_csc2_config_t::A2`

22.3.5.0.0.47.4 `float pxp_csc2_config_t::A3`

22.3.5.0.0.47.5 `float pxp_csc2_config_t::B1`

22.3.5.0.0.47.6 `float pxp_csc2_config_t::B2`

22.3.5.0.0.47.7 `float pxp_csc2_config_t::B3`

22.3.5.0.0.47.8 `float pxp_csc2_config_t::C1`

22.3.5.0.0.47.9 `float pxp_csc2_config_t::C2`

22.3.5.0.0.47.10 `float pxp_csc2_config_t::C3`

22.3.5.0.0.47.11 `uint16_t pxp_csc2_config_t::D1`

22.3.5.0.0.47.12 `uint16_t pxp_csc2_config_t::D2`

22.3.5.0.0.47.13 `uint16_t pxp_csc2_config_t::D3`

### 22.3.6 struct pxp\_lut\_config\_t

#### Data Fields

- `pxp_lut_lookup_mode_t lookupMode`  
*Look up mode.*
- `pxp_lut_out_mode_t outMode`  
*Out mode.*
- `uint32_t cfaValue`  
*The CFA value used when look up mode is [kPXP\\_LutOutRGBW4444CFA](#).*

## Data Structure Documentation

### 22.3.6.0.0.48 Field Documentation

22.3.6.0.0.48.1 `pxp_lut_lookup_mode_t ppx_lut_config_t::lookupMode`

22.3.6.0.0.48.2 `pxp_lut_out_mode_t ppx_lut_config_t::outMode`

22.3.6.0.0.48.3 `uint32_t ppx_lut_config_t::cfaValue`

### 22.3.7 `struct ppx_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.*

### 22.3.7.0.0.49 Field Documentation

22.3.7.0.0.49.1 `uint32_t ppx_dither_final_lut_data_t::data_3_0`

Data 0 is the least significant byte.

22.3.7.0.0.49.2 `uint32_t ppx_dither_final_lut_data_t::data_7_4`

Data 4 is the least significant byte.

22.3.7.0.0.49.3 `uint32_t ppx_dither_final_lut_data_t::data_11_8`

Data 8 is the least significant byte.

22.3.7.0.0.49.4 `uint32_t ppx_dither_final_lut_data_t::data_15_12`

Data 12 is the least significant byte.

### 22.3.8 `struct ppx_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.*

### 22.3.8.0.0.50 Field Documentation

**22.3.8.0.0.50.1 `uint32_t ppx_dither_config_t::enableDither0`**

**22.3.8.0.0.50.2 `uint32_t ppx_dither_config_t::enableDither1`**

**22.3.8.0.0.50.3 `uint32_t ppx_dither_config_t::enableDither2`**

**22.3.8.0.0.50.4 `uint32_t ppx_dither_config_t::ditherMode0`**

See [\\_pxp\\_dither\\_mode](#).

**22.3.8.0.0.50.5 `uint32_t ppx_dither_config_t::ditherMode1`**

See [\\_pxp\\_dither\\_mode](#).

**22.3.8.0.0.50.6 `uint32_t ppx_dither_config_t::ditherMode2`**

See [\\_pxp\\_dither\\_mode](#).

**22.3.8.0.0.50.7 `uint32_t ppx_dither_config_t::quantBitNum`**

**22.3.8.0.0.50.8 `uint32_t ppx_dither_config_t::lutMode`**

This must be set to [kPXP\\_DitherLutOff](#) if any dither engine uses [kPXP\\_DitherOrdered](#) mode.

## Enumeration Type Documentation

22.3.8.0.0.50.9 `uint32_t ppx_dither_config_t::idxMatrixSize0`  
22.3.8.0.0.50.10 `uint32_t ppx_dither_config_t::idxMatrixSize1`  
22.3.8.0.0.50.11 `uint32_t ppx_dither_config_t::idxMatrixSize2`  
22.3.8.0.0.50.12 `uint32_t ppx_dither_config_t::enableFinalLut`

## 22.4 Enumeration Type Documentation

### 22.4.1 enum \_ppx\_interrupt\_enable

Enumerator

***kPXP\_CommandLoadInterruptEnable*** Interrupt to show that the command set by [PXP\\_SetNextCommand](#) has been loaded.  
***kPXP\_CompleteInterruptEnable*** PXP process completed.  
***kPXP\_LutDmaLoadInterruptEnable*** The LUT table has been loaded by DMA.

### 22.4.2 enum \_ppx\_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\_LutDmaLoadFlag*** The LUT table has been loaded by DMA.  
***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.

### 22.4.3 enum ppx\_flip\_mode\_t

Enumerator

***kPXP\_FlipDisable*** Flip disable.  
***kPXP\_FlipHorizontal*** Horizontal flip.  
***kPXP\_FlipVertical*** Vertical flip.  
***kPXP\_FlipBoth*** Flip both directions.

#### 22.4.4 enum pxp\_rotate\_position\_t

Enumerator

- kPXP\_RotateOutputBuffer* Rotate the output buffer.
- kPXP\_RotateProcessSurface* Rotate the process surface.

#### 22.4.5 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.

#### 22.4.6 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.

#### 22.4.7 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\_OutputPixelFormatYUV1P444* 32-bit pixels (1-plane XYUV unpacked).
- kPXP\_OutputPixelFormatUYVY1P422* 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)
- kPXP\_OutputPixelFormatVYUY1P422* 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)

## Enumeration Type Documentation

|                                       |                                              |
|---------------------------------------|----------------------------------------------|
| <i>kPXP_OutputPixelFormatYUV2P422</i> | 16-bit pixels (2-plane UV interleaved bytes) |
| <i>kPXP_OutputPixelFormatYUV2P420</i> | 16-bit pixels (2-plane UV)                   |
| <i>kPXP_OutputPixelFormatYVU2P422</i> | 16-bit pixels (2-plane VU interleaved bytes) |
| <i>kPXP_OutputPixelFormatYVU2P420</i> | 16-bit pixels (2-plane VU)                   |

### 22.4.8 enum pxp\_ps\_pixel\_format\_t

Enumerator

|                                    |                                                            |
|------------------------------------|------------------------------------------------------------|
| <i>kPXP_PsPixelFormatRGB888</i>    | 32-bit pixels without alpha (unpacked 24-bit format)       |
| <i>kPXP_PsPixelFormatRGB555</i>    | 16-bit pixels without alpha.                               |
| <i>kPXP_PsPixelFormatRGB444</i>    | 16-bit pixels without alpha.                               |
| <i>kPXP_PsPixelFormatRGB565</i>    | 16-bit pixels without alpha.                               |
| <i>kPXP_PsPixelFormatYUVIP444</i>  | 32-bit pixels (1-plane XYUV unpacked).                     |
| <i>kPXP_PsPixelFormatUYVY1P422</i> | 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)      |
| <i>kPXP_PsPixelFormatVYUY1P422</i> | 16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)      |
| <i>kPXP_PsPixelFormatY8</i>        | 8-bit monochrome pixels (1-plane Y luma output)            |
| <i>kPXP_PsPixelFormatY4</i>        | 4-bit monochrome pixels (1-plane Y luma, 4 bit truncation) |
| <i>kPXP_PsPixelFormatYUV2P422</i>  | 16-bit pixels (2-plane UV interleaved bytes)               |
| <i>kPXP_PsPixelFormatYUV2P420</i>  | 16-bit pixels (2-plane UV)                                 |
| <i>kPXP_PsPixelFormatYVU2P422</i>  | 16-bit pixels (2-plane VU interleaved bytes)               |
| <i>kPXP_PsPixelFormatYVU2P420</i>  | 16-bit pixels (2-plane VU)                                 |
| <i>kPXP_PsPixelFormatYVU422</i>    | 16-bit pixels (3-plane)                                    |
| <i>kPXP_PsPixelFormatYVU420</i>    | 16-bit pixels (3-plane)                                    |

### 22.4.9 enum pxp\_as\_pixel\_format\_t

Enumerator

|                                   |                                                      |
|-----------------------------------|------------------------------------------------------|
| <i>kPXP_AsPixelFormatARGB8888</i> | 32-bit pixels with alpha.                            |
| <i>kPXP_AsPixelFormatRGB888</i>   | 32-bit pixels without alpha (unpacked 24-bit format) |
| <i>kPXP_AsPixelFormatARGB1555</i> | 16-bit pixels with alpha.                            |
| <i>kPXP_AsPixelFormatARGB4444</i> | 16-bit pixels with alpha.                            |
| <i>kPXP_AsPixelFormatRGB555</i>   | 16-bit pixels without alpha.                         |
| <i>kPXP_AsPixelFormatRGB444</i>   | 16-bit pixels without alpha.                         |
| <i>kPXP_AsPixelFormatRGB565</i>   | 16-bit pixels without alpha.                         |

### 22.4.10 enum pxp\_alpha\_mode\_t

Enumerator

|                           |                                                             |
|---------------------------|-------------------------------------------------------------|
| <i>kPXP_AlphaEmbedded</i> | 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 to 200, user defined alpha set to 100, then the result alpha is  $200 * 100 / 255$ .

***kPXP\_AlphaRop*** Raster operation.

#### 22.4.11 enum ppx\_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.

#### 22.4.12 enum ppx\_block\_size\_t

Enumerator

***kPXP\_BlockSize8*** Process 8x8 pixel blocks.

***kPXP\_BlockSize16*** Process 16x16 pixel blocks.

#### 22.4.13 enum ppx\_csc1\_mode\_t

Enumerator

***kPXP\_Csc1YUV2RGB*** YUV to RGB.

***kPXP\_Csc1YCbCr2RGB*** YCbCr to RGB.

## Enumeration Type Documentation

### 22.4.14 enum ppx\_csc2\_mode\_t

Enumerator

*kPPX\_Csc2YUV2RGB* YUV to RGB.  
*kPPX\_Csc2YCbCr2RGB* YCbCr to RGB.  
*kPPX\_Csc2RGB2YUV* RGB to YUV.  
*kPPX\_Csc2RGB2YCbCr* RGB to YCbCr.

### 22.4.15 enum ppx\_lut\_lookup\_mode\_t

Enumerator

*kPPX\_LutCacheRGB565* LUT ADDR = R[7:3],G[7:2],B[7:3]. Use all 16KB of LUT for indirect cached 128KB lookup.  
*kPPX\_LutDirectY8* LUT ADDR = 16'b0,Y[7:0]. Use the first 256 bytes of LUT. Only third data path byte is transformed.  
*kPPX\_LutDirectRGB444* LUT ADDR = R[7:4],G[7:4],B[7:4]. Use one 8KB bank of LUT selected by [PPX\\_Select8kLutBank](#).  
*kPPX\_LutDirectRGB454* LUT ADDR = R[7:4],G[7:3],B[7:4]. Use all 16KB of LUT.

### 22.4.16 enum ppx\_lut\_out\_mode\_t

Enumerator

*kPPX\_LutOutY8* R/Y byte lane 2 lookup, bytes 1,0 bypassed.  
*kPPX\_LutOutRGBW4444CFA* Byte lane 2 = CFA\_Y8, byte lane 1,0 = RGBW4444.  
*kPPX\_LutOutRGB888* RGB565->RGB888 conversion for Gamma correction.

### 22.4.17 enum ppx\_lut\_8k\_bank\_t

Enumerator

*kPPX\_Lut8kBank0* The first 8K bank used.  
*kPPX\_Lut8kBank1* The second 8K bank used.

### 22.4.18 enum ppx\_ram\_t

Enumerator

*kPPX\_RamDither0Lut* Dither 0 LUT memory.

***kPXP\_RamDither1Lut*** Dither 1 LUT memory.

***kPXP\_RamDither2Lut*** Dither 2 LUT memory.

#### 22.4.19 enum \_pxp\_dither\_mode

Enumerator

***kPXP\_DitherPassThrough*** Pass through, no dither.

***kPXP\_DitherOrdered*** Ordered dither.

***kPXP\_DitherQuantOnly*** No dithering, only quantization.

#### 22.4.20 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.

#### 22.4.21 enum \_pxp\_dither\_matrix\_size

Enumerator

***kPXP\_DitherMatrix8*** The dither index matrix is 8x8.

***kPXP\_DitherMatrix16*** The dither index matrix is 16x16.

### 22.5 Function Documentation

#### 22.5.1 void PXP\_Init ( PXP\_Type \* *base* )

This function enables the PXP peripheral clock, and resets the PXP registers to default status.

Parameters

---

## Function Documentation

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

### 22.5.2 void PXP\_Deinit ( PXP\_Type \* *base* )

This function disables the PXP peripheral clock.

Parameters

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

### 22.5.3 void PXP\_Reset ( PXP\_Type \* *base* )

This function resets the PXP peripheral registers to default status.

Parameters

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

### 22.5.4 static void PXP\_Start ( PXP\_Type \* *base* ) [inline], [static]

Start PXP process using current configuration.

Parameters

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

### 22.5.5 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. |
|---------------|-----------------------------------|

### 22.5.6 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. |

### 22.5.7 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 selected.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | PXP peripheral base address. |
| <i>size</i> | The pixel block size.        |

### 22.5.8 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))
{
    ...
}
```

## Function Documentation

Parameters

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

Returns

PXP status flags which are OR'ed by the enumerators in the `_pxp_flags`.

### 22.5.9 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 <code>_pxp_flags</code> . |

### 22.5.10 static uint8\_t PXP\_GetAxiErrorId ( **PXP\_Type** \* *base*, **uint8\_t** *axiIndex* ) [inline], [static]

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.

### 22.5.11 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);
```

## Function Documentation

Parameters

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>base</i> | PXP peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_pxp_interrupt_enable</a> . |

### 22.5.12 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 <a href="#">_pxp_interrupt_enable</a> . |

### 22.5.13 void PXP\_SetAlphaSurfaceBufferConfig ( PXP\_Type \* *base*, const ppx\_as\_buffer\_config\_t \* *config* )

Parameters

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

### 22.5.14 void PXP\_SetAlphaSurfaceBlendConfig ( PXP\_Type \* *base*, const ppx\_as\_blend\_config\_t \* *config* )

Parameters

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

### 22.5.15 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

### 22.5.16 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. |

### 22.5.17 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.  |

## Function Documentation

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

**22.5.18 static void PXP\_SetProcessSurfaceBackGroundColor ( PXP\_Type \* *base*, uint32\_t *backGroundColor* ) [inline], [static]**

Parameters

|                         |                                      |
|-------------------------|--------------------------------------|
| <i>base</i>             | PXP peripheral base address.         |
| <i>backGround-Color</i> | Pixel value of the background color. |

**22.5.19 void PXP\_SetProcessSurfaceBufferConfig ( PXP\_Type \* *base*, const pxp\_ps\_buffer\_config\_t \* *config* )**

Parameters

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

**22.5.20 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. |

**22.5.21 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. |

**22.5.22 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 0xFF-FFFF 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.        |

**22.5.23 void PXP\_SetOutputBufferConfig ( *PXP\_Type* \* *base*, *const ppxp\_output\_buffer\_config\_t* \* *config* )**

Parameters

## Function Documentation

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

### 22.5.24 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.             |

### 22.5.25 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. |

### 22.5.26 static void PXP\_SetRotateConfig ( PXP\_Type \* *base*, ppx\_rotate\_position\_t *position*, ppx\_rotate\_degree\_t *degree*, ppx\_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 rotated 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.

### 22.5.27 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 ppxp_command1[48];
uint32_t ppxp_command2[48];

// Prepare the register values.
ppxp_command1[0] = ...;
ppxp_command1[1] = ...;
// ...
ppxp_command2[0] = ...;
ppxp_command2[1] = ...;
// ...

// Make sure no new command pending.
while (PXP_IsNextCommandPending(PXP))
{
}

// Set new operation.
PXP_SetNextCommand(PXP, ppxp_command1);

// Wait for new command loaded. Here could check @ref kPXP_CommandLoadFlag too.
while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, ppxp_command2);

```

## Function Documentation

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>commandAddr</i> | Address of the new command.  |

**22.5.28 static bool PXP\_IsNextCommandPending ( PXP\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

True is pending, false is not.

**22.5.29 static void PXP\_CancelNextCommand ( PXP\_Type \* *base* ) [inline],  
[static]**

Parameters

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

**22.5.30 void PXP\_SetCsc2Config ( PXP\_Type \* *base*, const ppx\_csc2\_config\_t \*  
*config* )**

The CSC2 module receives pixels in any color space and can convert the pixels into any of RGB, YUV, or YCbCr color spaces. The output pixels are passed onto the LUT and rotation engine for further processing

Parameters

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

**22.5.31 static void PXP\_EnableCsc2 ( PXP\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

### 22.5.32 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.         |

### 22.5.33 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. |

### 22.5.34 void PXP\_SetLutConfig ( PXP\_Type \* *base*, const pxp\_lut\_config\_t \* *config* )

The lookup table (LUT) is used to modify pixels in a manner that is not linear and that cannot be achieved by the color space conversion modules. To setup the LUT, the complete workflow is:

1. Use [PXP\\_SetLutConfig](#) to set the configuration, such as the lookup mode.
2. Use [PXP\\_LoadLutTable](#) to load the lookup table to PXP.
3. Use [PXP\\_EnableLut](#) to enable the function.

Parameters

---

## Function Documentation

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

**22.5.35 status\_t PXP\_LoadLutTable ( PXP\_Type \* *base*, ppx\_lut\_lookup\_mode\_t *lookupMode*, uint32\_t *bytesNum*, uint32\_t *memAddr*, uint16\_t *lutStartAddr* )**

If lookup mode is DIRECT mode, this function loads *bytesNum* of values from the address *memAddr* into PXP LUT address *lutStartAddr*. So this function allows only update part of the PXP LUT.

If lookup mode is CACHE mode, this function sets the new address to *memAddr* and invalid the PXP LUT cache.

Parameters

|                     |                                                                                                                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | PXP peripheral base address.                                                                                                                                                                                                                       |
| <i>lookupMode</i>   | Which lookup mode is used. Note that this parameter is only used to distinguish DIRECT mode and CACHE mode, it does not change the register value PXP_LUT-CTRL[LOOKUP_MODE]. To change that value, use function <a href="#">PXP_SetLutConfig</a> . |
| <i>bytesNum</i>     | How many bytes to set. This value must be divisible by 8.                                                                                                                                                                                          |
| <i>memAddr</i>      | Address of look up table to set.                                                                                                                                                                                                                   |
| <i>lutStartAddr</i> | The LUT value will be loaded to LUT from index <i>lutAddr</i> . It should be 8 bytes aligned.                                                                                                                                                      |

Return values

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kStatus_Success</i>         | Load successfully.                  |
| <i>kStatus_InvalidArgument</i> | Failed because of invalid argument. |

**22.5.36 static void PXP\_EnableLut ( PXP\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

22.5.37 **static void PXP\_Select8kLutBank( PXP\_Type \* *base*, pxp\_lut\_8k\_bank\_t *bank* ) [inline], [static]**

## Function Documentation

### Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | PXP peripheral base address. |
| <i>bank</i> | The bank to select.          |

# Chapter 23

## QSPI: Quad Serial Peripheral Interface Driver

### 23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Quad Serial Peripheral Interface (QSPI) module of MCUXpresso SDK devices.

QSPI driver includes functional APIs and EDMA transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for QSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the QSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. QSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `qspi_handle_t` as the first parameter. Initialize the handle by calling the [QSPI\\_TransferTxCreateHandleEDMA\(\)](#) or [QSPI\\_TransferRxCreateHandleEDMA\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [QSPI\\_TransferSendEDMA\(\)](#) and [QSPI\\_TransferReceiveEDMA\(\)](#) set up EDMA for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_QSPI_Idle` status.

### Modules

- [QSPI eDMA Driver](#)

### Data Structures

- struct `qspi_dqs_config_t`  
*DQS configure features.* [More...](#)
- struct `qspi_flash_timing_t`  
*Flash timing configuration.* [More...](#)
- struct `qspi_config_t`  
*QSPI configuration structure.* [More...](#)
- struct `qspi_flash_config_t`  
*External flash configuration items.* [More...](#)
- struct `qspi_transfer_t`  
*Transfer structure for QSPI.* [More...](#)

### Macros

- #define `QSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)`

## Overview

- Macro functions for LUT table.
  - `#define QSPI_CMD (0x1U)`  
*Macro for QSPI LUT command.*
  - `#define QSPI_PAD_1 (0x0U)`  
*Macro for QSPI PAD.*

## Enumerations

- enum `_status_t`{  
  `kStatus_QSPI_Idle` = MAKE\_STATUS(kStatusGroup\_QSPI, 0),  
  `kStatus_QSPI_Busy` = MAKE\_STATUS(kStatusGroup\_QSPI, 1),  
  `kStatus_QSPI_Error` = MAKE\_STATUS(kStatusGroup\_QSPI, 2) }  
  
*Status structure of QSPI.*
- enum `qspi_read_area_t`{  
  `kQSPI_ReadAHB` = 0x0U,  
  `kQSPI_ReadIP` }  
  
*QSPI read data area, from IP FIFO or AHB buffer.*
- enum `qspi_command_seq_t`{  
  `kQSPI_IPSeq` = QuadSPI\_SPTRCLR\_IPPTRC\_MASK,  
  `kQSPI_BufferSeq` = QuadSPI\_SPTRCLR\_BFPTRC\_MASK }  
  
*QSPI command sequence type.*
- enum `qspi_fifo_t`{  
  `kQSPI_TxFifo` = QuadSPI\_MCR\_CLR\_TXF\_MASK,  
  `kQSPI_RxFifo` = QuadSPI\_MCR\_CLR\_RXF\_MASK,  
  `kQSPI_AllFifo` = QuadSPI\_MCR\_CLR\_TXF\_MASK | QuadSPI\_MCR\_CLR\_RXF\_MASK }  
  
*QSPI buffer type.*
- enum `qspi_endianness_t`{  
  `kQSPI_64BigEndian` = 0x0U,  
  `kQSPI_32LittleEndian`,  
  `kQSPI_32BigEndian`,  
  `kQSPI_64LittleEndian` }  
  
*QSPI transfer endianess.*
- enum `_qspi_error_flags`{  
  `kQSPI_DataLearningFail` = QuadSPI\_FR\_DLPFF\_MASK,  
  `kQSPI_TxBufferFill` = QuadSPI\_FR\_TBFF\_MASK,  
  `kQSPI_TxBufferUnderrun` = QuadSPI\_FR\_TBUF\_MASK,  
  `kQSPI_IllegalInstruction` = QuadSPI\_FR\_ILLINE\_MASK,  
  `kQSPI_RxBufferOverflow` = QuadSPI\_FR\_RBOF\_MASK,  
  `kQSPI_RxBufferDrain` = QuadSPI\_FR\_RBDF\_MASK,  
  `kQSPI_AHBSequenceError` = QuadSPI\_FR\_ABSEF\_MASK,  
  `kQSPI_AHBBufferOverflow` = QuadSPI\_FR\_ABOF\_MASK,  
  `kQSPI_IPCommandUsageError` = QuadSPI\_FR\_IUEF\_MASK,  
  `kQSPI_IPCommandTriggerDuringAHBAccess` = QuadSPI\_FR\_IPAEF\_MASK,  
  `kQSPI_IPCommandTriggerDuringIPAccess` = QuadSPI\_FR\_IPIEF\_MASK,  
  `kQSPI_IPCommandTriggerDuringAHBGrant` = QuadSPI\_FR\_IPGEF\_MASK,  
  `kQSPI_IPCommandTransactionFinished` = QuadSPI\_FR\_TFF\_MASK,  
  `kQSPI_FlagAll` = 0x8C83F8D1U }

*QSPI error flags.*

- enum `_qspi_flags` {
   
kQSPI\_DataLearningSamplePoint = QuadSPI\_SR\_DLPSMP\_MASK,
   
kQSPI\_TxBufferFull = QuadSPI\_SR\_TXFULL\_MASK,
   
kQSPI\_TxBufferEnoughData = QuadSPI\_SR\_TXEDA\_MASK,
   
kQSPI\_RxDMA = QuadSPI\_SR\_RXDMA\_MASK,
   
kQSPI\_RxBufferFull = QuadSPI\_SR\_RXFULL\_MASK,
   
kQSPI\_RxWatermark = QuadSPI\_SR\_RXWE\_MASK,
   
kQSPI\_AHB3BufferFull = QuadSPI\_SR\_AHB3FUL\_MASK,
   
kQSPI\_AHB2BufferFull = QuadSPI\_SR\_AHB2FUL\_MASK,
   
kQSPI\_AHB1BufferFull = QuadSPI\_SR\_AHB1FUL\_MASK,
   
kQSPI\_AHB0BufferFull = QuadSPI\_SR\_AHB0FUL\_MASK,
   
kQSPI\_AHB3BufferNotEmpty = QuadSPI\_SR\_AHB3NE\_MASK,
   
kQSPI\_AHB2BufferNotEmpty = QuadSPI\_SR\_AHB2NE\_MASK,
   
kQSPI\_AHB1BufferNotEmpty = QuadSPI\_SR\_AHB1NE\_MASK,
   
kQSPI\_AHB0BufferNotEmpty = QuadSPI\_SR\_AHB0NE\_MASK,
   
kQSPI\_AHBTransactionPending = QuadSPI\_SR\_AHBTRN\_MASK,
   
kQSPI\_AHBCCommandPriorityGranted = QuadSPI\_SR\_AHBGNT\_MASK,
   
kQSPI\_AHBAccess = QuadSPI\_SR\_AHB\_ACC\_MASK,
   
kQSPI\_IPAccess = QuadSPI\_SR\_IP\_ACC\_MASK,
   
kQSPI\_Busy = QuadSPI\_SR\_BUSY\_MASK,
   
kQSPI\_StateAll = 0xEF897FE7U }

*QSPI state bit.*

- enum `_qspi_interrupt_enable` {
   
kQSPI\_DataLearningFailInterruptEnable,
   
kQSPI\_TxBufferFillInterruptEnable = QuadSPI\_RSER\_TBFIE\_MASK,
   
kQSPI\_TxBufferUnderrunInterruptEnable = QuadSPI\_RSER\_TBUIE\_MASK,
   
kQSPI\_IllegalInstructionInterruptEnable,
   
kQSPI\_RxBufferOverflowInterruptEnable = QuadSPI\_RSER\_RBOIE\_MASK,
   
kQSPI\_RxBufferDrainInterruptEnable = QuadSPI\_RSER\_RBDIE\_MASK,
   
kQSPI\_AHBSequenceErrorInterruptEnable = QuadSPI\_RSER\_ABSEIE\_MASK,
   
kQSPI\_AHBBufferOverflowInterruptEnable = QuadSPI\_RSER\_ABOIE\_MASK,
   
kQSPI\_IPCommandUsageErrorInterruptEnable = QuadSPI\_RSER\_IUEIE\_MASK,
   
kQSPI\_IPCommandTriggerDuringAHBAccessInterruptEnable,
   
kQSPI\_IPCommandTriggerDuringIPAccessInterruptEnable,
   
kQSPI\_IPCommandTriggerDuringAHBGrantInterruptEnable,
   
kQSPI\_IPCommandTransactionFinishedInterruptEnable,
   
kQSPI\_AllInterruptEnable = 0x8C83F8D1U }

*QSPI interrupt enable.*

- enum `_qspi_dma_enable` { kQSPI\_RxBufferDrainDMAEnable = QuadSPI\_RSER\_RBDDE\_MASK }
- QSPI DMA request flag.*
- enum `qspi_dqs_phrase_shift_t` {
   
kQSPI\_DQSNoPhraseShift = 0x0U,
   
kQSPI\_DQSPhraseShift45Degree,
   
kQSPI\_DQSPhraseShift90Degree,

## Overview

```
kQSPI_DQSPhraseShift135Degree }
```

*Phrase shift number for DQS mode.*

## Driver version

- #define **FSL\_QSPI\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 2))  
*I2C driver version 2.0.2.*

## Initialization and deinitialization

- void **QSPI\_Init** (QuadSPI\_Type \*base, **qspi\_config\_t** \*config, uint32\_t srcClock\_Hz)  
*Initializes the QSPI module and internal state.*
- void **QSPI\_GetDefaultQspiConfig** (**qspi\_config\_t** \*config)  
*Gets default settings for QSPI.*
- void **QSPI\_Deinit** (QuadSPI\_Type \*base)  
*Deinitializes the QSPI module.*
- void **QSPI\_SetFlashConfig** (QuadSPI\_Type \*base, **qspi\_flash\_config\_t** \*config)  
*Configures the serial flash parameter.*
- void **QSPI\_SoftwareReset** (QuadSPI\_Type \*base)  
*Software reset for the QSPI logic.*
- static void **QSPI\_Enable** (QuadSPI\_Type \*base, bool enable)  
*Enables or disables the QSPI module.*

## Status

- static uint32\_t **QSPI\_GetStatusFlags** (QuadSPI\_Type \*base)  
*Gets the state value of QSPI.*
- static uint32\_t **QSPI\_GetErrorStatusFlags** (QuadSPI\_Type \*base)  
*Gets QSPI error status flags.*
- static void **QSPI\_ClearErrorFlag** (QuadSPI\_Type \*base, uint32\_t mask)  
*Clears the QSPI error flags.*

## Interrupts

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

## DMA Control

- static void **QSPI\_EnableDMA** (QuadSPI\_Type \*base, uint32\_t mask, bool enable)  
*Enables the QSPI DMA source.*
- static uint32\_t **QSPI\_GetTxDataRegisterAddress** (QuadSPI\_Type \*base)  
*Gets the Tx data register address.*
- uint32\_t **QSPI\_GetRxDataRegisterAddress** (QuadSPI\_Type \*base)  
*Gets the Rx data register address used for DMA operation.*

## Bus Operations

- static void [QSPI\\_SetIPCommandAddress](#) (QuadSPI\_Type \*base, uint32\_t addr)  
*Sets the IP command address.*
- static void [QSPI\\_SetIPCommandSize](#) (QuadSPI\_Type \*base, uint32\_t size)  
*Sets the IP command size.*
- void [QSPI\\_ExecuteIPCommand](#) (QuadSPI\_Type \*base, uint32\_t index)  
*Executes IP commands located in LUT table.*
- void [QSPI\\_ExecuteAHBCommand](#) (QuadSPI\_Type \*base, uint32\_t index)  
*Executes AHB commands located in LUT table.*
- static void [QSPI\\_EnableIPParallelMode](#) (QuadSPI\_Type \*base, bool enable)  
*Enables/disables the QSPI IP command parallel mode.*
- static void [QSPI\\_EnableAHBParallelMode](#) (QuadSPI\_Type \*base, bool enable)  
*Enables/disables the QSPI AHB command parallel mode.*
- void [QSPI\\_UpdateLUT](#) (QuadSPI\_Type \*base, uint32\_t index, uint32\_t \*cmd)  
*Updates the LUT table.*
- static void [QSPI\\_ClearFifo](#) (QuadSPI\_Type \*base, uint32\_t mask)  
*Clears the QSPI FIFO logic.*
- static void [QSPI\\_ClearCommandSequence](#) (QuadSPI\_Type \*base, [qspi\\_command\\_seq\\_t](#) seq)  
*@ brief Clears the command sequence for the IP/buffer command.*
- static void [QSPI\\_EnableDDRMode](#) (QuadSPI\_Type \*base, bool enable)  
*Enable or disable DDR mode.*
- void [QSPI\\_SetReadDataArea](#) (QuadSPI\_Type \*base, [qspi\\_read\\_area\\_t](#) area)  
*@ brief Set the RX buffer readout area.*
- void [QSPI\\_WriteBlocking](#) (QuadSPI\_Type \*base, uint32\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes using a blocking method.*
- static void [QSPI\\_WriteData](#) (QuadSPI\_Type \*base, uint32\_t data)  
*Writes data into FIFO.*
- void [QSPI\\_ReadBlocking](#) (QuadSPI\_Type \*base, uint32\_t \*buffer, size\_t size)  
*Receives a buffer of data bytes using a blocking method.*
- uint32\_t [QSPI\\_ReadData](#) (QuadSPI\_Type \*base)  
*Receives data from data FIFO.*

## Transactional

- static void [QSPI\\_TransferSendBlocking](#) (QuadSPI\_Type \*base, [qspi\\_transfer\\_t](#) \*xfer)  
*Writes data to the QSPI transmit buffer.*
- static void [QSPI\\_TransferReceiveBlocking](#) (QuadSPI\_Type \*base, [qspi\\_transfer\\_t](#) \*xfer)  
*Reads data from the QSPI receive buffer in polling way.*

## 23.2 Data Structure Documentation

### 23.2.1 struct qspi\_dqs\_config\_t

#### Data Fields

- uint32\_t portADelayTapNum  
*Delay chain tap number selection for QSPI port A DQS.*
- uint32\_t portBDelayTapNum  
*Delay chain tap number selection for QSPI port B DQS.*

## Data Structure Documentation

- **qspi\_dqs\_phrase\_shift\_t shift**  
*Phase shift for internal DQS generation.*
- **bool enableDQSClkInverse**  
*Enable inverse clock for internal DQS generation.*
- **bool enableDQSPadLoopback**  
*Enable DQS loop back from DQS pad.*
- **bool enableDQSLoopback**  
*Enable DQS loop back.*

### 23.2.2 struct qspi\_flash\_timing\_t

#### Data Fields

- **uint32\_t dataHoldTime**  
*Serial flash data in hold time.*
- **uint32\_t CSHoldTime**  
*Serial flash CS hold time in terms of serial flash clock cycles.*
- **uint32\_t CSSetupTime**  
*Serial flash CS setup time in terms of serial flash clock cycles.*

### 23.2.3 struct qspi\_config\_t

#### Data Fields

- **uint32\_t clockSource**  
*Clock source for QSPI module.*
- **uint32\_t baudRate**  
*Serial flash clock baud rate.*
- **uint8\_t txWatermark**  
*QSPI transmit watermark value.*
- **uint8\_t rxWatermark**  
*QSPI receive watermark value.*
- **uint32\_t AHBbufferSize [FSL\_FEATURE\_QSPI\_AHB\_BUFFER\_COUNT]**  
*AHB buffer size.*
- **uint8\_t AHBbufferMaster [FSL\_FEATURE\_QSPI\_AHB\_BUFFER\_COUNT]**  
*AHB buffer master.*
- **bool enableAHBbuffer3AllMaster**  
*Is AHB buffer3 for all master.*
- **qspi\_read\_area\_t area**  
*Which area Rx data readout.*
- **bool enableQspi**  
*Enable QSPI after initialization.*

### 23.2.3.0.0.51 Field Documentation

23.2.3.0.0.51.1 `uint8_t qspi_config_t::rxWatermark`

23.2.3.0.0.51.2 `uint32_t qspi_config_t::AHBbufferSize[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]`

23.2.3.0.0.51.3 `uint8_t qspi_config_t::AHBbufferMaster[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]`

23.2.3.0.0.51.4 `bool qspi_config_t::enableAHBbuffer3AllMaster`

## 23.2.4 struct qspi\_flash\_config\_t

### Data Fields

- `uint32_t flashA1Size`  
*Flash A1 size.*
- `uint32_t flashA2Size`  
*Flash A2 size.*
- `uint32_t flashB1Size`  
*Flash B1 size.*
- `uint32_t flashB2Size`  
*Flash B2 size.*
- `uint32_t lookuptable [FSL_FEATURE_QSPI_LUT_DEPTH]`  
*Flash command in LUT.*
- `uint32_t CSHoldTime`  
*CS line hold time.*
- `uint32_t CSSetupTime`  
*CS line setup time.*
- `uint32_t columnSpace`  
*Column space size.*
- `uint32_t dataLearnValue`  
*Data Learn value if enable data learn.*
- `qspi_endianness_t endian`  
*Flash data endianness.*
- `bool enableWordAddress`  
*If enable word address.*

## Enumeration Type Documentation

### 23.2.4.0.0.52 Field Documentation

23.2.4.0.0.52.1 `qspi_endianness_t qspi_flash_config_t::endian`

23.2.4.0.0.52.2 `bool qspi_flash_config_t::enableWordAddress`

### 23.2.5 `struct qspi_transfer_t`

#### Data Fields

- `uint32_t * data`  
*Pointer to data to transmit.*
- `size_t dataSize`  
*Bytes to be transmit.*

### 23.3 Macro Definition Documentation

23.3.1 `#define FSL_QSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

### 23.4 Enumeration Type Documentation

#### 23.4.1 `enum _status_t`

Enumerator

`kStatus_QSPI_Idle` QSPI is in idle state.

`kStatus_QSPI_Busy` QSPI is busy.

`kStatus_QSPI_Error` Error occurred during QSPI transfer.

#### 23.4.2 `enum qspi_read_area_t`

Enumerator

`kQSPI_ReadAHB` QSPI read from AHB buffer.

`kQSPI_ReadIP` QSPI read from IP FIFO.

#### 23.4.3 `enum qspi_command_seq_t`

Enumerator

`kQSPI_IPSeq` IP command sequence.

`kQSPI_BufferSeq` Buffer command sequence.

### 23.4.4 enum qspi\_fifo\_t

Enumerator

- kQSPI\_TxFifo* QSPI Tx FIFO.
- kQSPI\_RxFifo* QSPI Rx FIFO.
- kQSPI\_AllFifo* QSPI all FIFO, including Tx and Rx.

### 23.4.5 enum qspi\_endianness\_t

Enumerator

- kQSPI\_64BigEndian* 64 bits big endian
- kQSPI\_32LittleEndian* 32 bit little endian
- kQSPI\_32BigEndian* 32 bit big endian
- kQSPI\_64LittleEndian* 64 bit little endian

### 23.4.6 enum \_qspi\_error\_flags

Enumerator

- kQSPI\_DataLearningFail* Data learning pattern failure flag.
- kQSPI\_TxBufferFill* Tx buffer fill flag.
- kQSPI\_TxBufferUnderrun* Tx buffer underrun flag.
- kQSPI\_IllegalInstruction* Illegal instruction error flag.
- kQSPI\_RxBufferOverflow* Rx buffer overflow flag.
- kQSPI\_RxBufferDrain* Rx buffer drain flag.
- kQSPI\_AHBSequenceError* AHB sequence error flag.
- kQSPI\_AHBBufferOverflow* AHB buffer overflow flag.
- kQSPI\_IPCommandUsageError* IP command usage error flag.
- kQSPI\_IPCommandTriggerDuringAHBAccess* IP command trigger during AHB access error.
- kQSPI\_IPCommandTriggerDuringIPAccess* IP command trigger cannot be executed.
- kQSPI\_IPCommandTriggerDuringAHBGrant* IP command trigger during AHB grant error.
- kQSPI\_IPCommandTransactionFinished* IP command transaction finished flag.
- kQSPI\_FlagAll* All error flag.

### 23.4.7 enum \_qspi\_flags

Enumerator

- kQSPI\_DataLearningSamplePoint* Data learning sample point.
- kQSPI\_TxBufferFull* Tx buffer full flag.

## Enumeration Type Documentation

*kQSPI\_TxBufferEnoughData* Tx buffer enough data available.  
*kQSPI\_RxDMA* Rx DMA is requesting or running.  
*kQSPI\_RxBufferFull* Rx buffer full.  
*kQSPI\_RxWatermark* Rx buffer watermark exceeded.  
*kQSPI\_AHB3BufferFull* AHB buffer 3 full.  
*kQSPI\_AHB2BufferFull* AHB buffer 2 full.  
*kQSPI\_AHB1BufferFull* AHB buffer 1 full.  
*kQSPI\_AHB0BufferFull* AHB buffer 0 full.  
*kQSPI\_AHB3BufferNotEmpty* AHB buffer 3 not empty.  
*kQSPI\_AHB2BufferNotEmpty* AHB buffer 2 not empty.  
*kQSPI\_AHB1BufferNotEmpty* AHB buffer 1 not empty.  
*kQSPI\_AHB0BufferNotEmpty* AHB buffer 0 not empty.  
*kQSPI\_AHBTxactionPending* AHB access transaction pending.  
*kQSPI\_AHBCmdPriorityGranted* AHB command priority granted.  
*kQSPI\_AHBAcces* AHB access.  
*kQSPI\_IPAccess* IP access.  
*kQSPI\_Busy* Module busy.  
*kQSPI\_StateAll* All flags.

### 23.4.8 enum \_qspi\_interrupt\_enable

Enumerator

*kQSPI\_DataLearningFailInterruptEnable* Data learning pattern failure interrupt enable.  
*kQSPI\_TxBufferFillInterruptEnable* Tx buffer fill interrupt enable.  
*kQSPI\_TxBufferUnderrunInterruptEnable* Tx buffer underrun interrupt enable.  
*kQSPI\_IllegalInstructionInterruptEnable* Illegal instruction error interrupt enable.  
*kQSPI\_RxBufferOverflowInterruptEnable* Rx buffer overflow interrupt enable.  
*kQSPI\_RxBufferDrainInterruptEnable* Rx buffer drain interrupt enable.  
*kQSPI\_AHBSquenceErrorInterruptEnable* AHB sequence error interrupt enable.  
*kQSPI\_AHBBufferOverflowInterruptEnable* AHB buffer overflow interrupt enable.  
*kQSPI\_IPCommandUsageErrorInterruptEnable* IP command usage error interrupt enable.  
*kQSPI\_IPCommandTriggerDuringAHBAccesInterruptEnable* IP command trigger during AHB access error.  
*kQSPI\_IPCommandTriggerDuringIPAccessInterruptEnable* IP command trigger cannot be executed.  
*kQSPI\_IPCommandTriggerDuringAHBGrantInterruptEnable* IP command trigger during AHB grant error.  
*kQSPI\_IPCommandTransactionFinishedInterruptEnable* IP command transaction finished interrupt enable.  
*kQSPI\_AllInterruptEnable* All error interrupt enable.

### 23.4.9 enum \_qspi\_dma\_enable

Enumerator

*kQSPI\_RxBufferDrainDMAEnable* Rx buffer drain DMA.

### 23.4.10 enum qspi\_dqs\_phrase\_shift\_t

Enumerator

*kQSPI\_DQSNoPhraseShift* No phase shift.

*kQSPI\_DQSPhraseShift45Degree* Select 45 degree phase shift.

*kQSPI\_DQSPhraseShift90Degree* Select 90 degree phase shift.

*kQSPI\_DQSPhraseShift135Degree* Select 135 degree phase shift.

## 23.5 Function Documentation

### 23.5.1 void QSPI\_Init ( QuadSPI\_Type \* *base*, qspi\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

This function enables the clock for QSPI and also configures the QSPI with the input configure parameters. Users should call this function before any QSPI operations.

Parameters

|                    |                                    |
|--------------------|------------------------------------|
| <i>base</i>        | Pointer to QuadSPI Type.           |
| <i>config</i>      | QSPI configure structure.          |
| <i>srcClock_Hz</i> | QSPI source clock frequency in Hz. |

### 23.5.2 void QSPI\_GetDefaultQspiConfig ( qspi\_config\_t \* *config* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>config</i> | QSPI configuration structure. |
|---------------|-------------------------------|

### 23.5.3 void QSPI\_Deinit ( QuadSPI\_Type \* *base* )

Clears the QSPI state and QSPI module registers.

## Function Documentation

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
|-------------|--------------------------|

### 23.5.4 void QSPI\_SetFlashConfig ( QuadSPI\_Type \* *base*, qspi\_flash\_config\_t \* *config* )

This function configures the serial flash relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the QSPI features.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.        |
| <i>config</i> | Flash configuration parameters. |

### 23.5.5 void QSPI\_SoftwareReset ( QuadSPI\_Type \* *base* )

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
|-------------|--------------------------|

### 23.5.6 static void QSPI\_Enable ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                     |
| <i>enable</i> | True means enable QSPI, false means disable. |

### 23.5.7 static uint32\_t QSPI\_GetStatusFlags ( QuadSPI\_Type \* *base* ) [inline], [static]

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
|-------------|--------------------------|

Returns

status flag, use status flag to AND `_qspi_flags` could get the related status.

### 23.5.8 static uint32\_t QSPI\_GetErrorStatusFlags ( QuadSPI\_Type \* *base* ) [inline], [static]

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
|-------------|--------------------------|

Returns

status flag, use status flag to AND `_qspi_error_flags` could get the related status.

### 23.5.9 static void QSPI\_ClearErrorFlag ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                           |
|-------------|-------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to QuadSPI Type.                                                                  |
| <i>mask</i> | Which kind of QSPI flags to be cleared, a combination of <code>_qspi_error_flags</code> . |

### 23.5.10 static void QSPI\_EnableInterrupts ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
|-------------|--------------------------|

## Function Documentation

|             |                        |
|-------------|------------------------|
| <i>mask</i> | QSPI interrupt source. |
|-------------|------------------------|

**23.5.11 static void QSPI\_DisableInterrupts ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
| <i>mask</i> | QSPI interrupt source.   |

**23.5.12 static void QSPI\_EnableDMA ( QuadSPI\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                    |
| <i>mask</i>   | QSPI DMA source.                            |
| <i>enable</i> | True means enable DMA, false means disable. |

**23.5.13 static uint32\_t QSPI\_GetTxDataRegisterAddress ( QuadSPI\_Type \* *base* ) [inline], [static]**

It is used for DMA operation.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
|-------------|--------------------------|

Returns

QSPI Tx data register address.

**23.5.14 uint32\_t QSPI\_GetRxDataRegisterAddress ( QuadSPI\_Type \* *base* )**

This function returns the Rx data register address or Rx buffer address according to the Rx read area settings.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
|-------------|--------------------------|

Returns

QSPI Rx data register address.

### 23.5.15 static void QSPI\_SetIPCommandAddress ( QuadSPI\_Type \* *base*, uint32\_t *addr* ) [inline], [static]

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
| <i>addr</i> | IP command address.      |

### 23.5.16 static void QSPI\_SetIPCommandSize ( QuadSPI\_Type \* *base*, uint32\_t *size* ) [inline], [static]

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
| <i>size</i> | IP command size.         |

### 23.5.17 void QSPI\_ExecuteIPCommand ( QuadSPI\_Type \* *base*, uint32\_t *index* )

Parameters

|              |                                              |
|--------------|----------------------------------------------|
| <i>base</i>  | Pointer to QuadSPI Type.                     |
| <i>index</i> | IP command located in which LUT table index. |

### 23.5.18 void QSPI\_ExecuteAHBCommand ( QuadSPI\_Type \* *base*, uint32\_t *index* )

## Function Documentation

Parameters

|              |                                               |
|--------------|-----------------------------------------------|
| <i>base</i>  | Pointer to QuadSPI Type.                      |
| <i>index</i> | AHB command located in which LUT table index. |

**23.5.19 static void QSPI\_EnableIPParallelMode ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                                            |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

**23.5.20 static void QSPI\_EnableAHBParallelMode ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                                            |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

**23.5.21 void QSPI\_UpdateLUT ( QuadSPI\_Type \* *base*, uint32\_t *index*, uint32\_t \* *cmd* )**

Parameters

|              |                                                                            |
|--------------|----------------------------------------------------------------------------|
| <i>base</i>  | Pointer to QuadSPI Type.                                                   |
| <i>index</i> | Which LUT index needs to be located. It should be an integer divided by 4. |
| <i>cmd</i>   | Command sequence array.                                                    |

**23.5.22 static void QSPI\_ClearFifo ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Pointer to QuadSPI Type.               |
| <i>mask</i> | Which kind of QSPI FIFO to be cleared. |

### 23.5.23 static void QSPI\_ClearCommandSequence ( QuadSPI\_Type \* *base*, qspi\_command\_seq\_t *seq* ) [inline], [static]

This function can reset the command sequence.

Parameters

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| <i>base</i> | QSPI base address.                                                        |
| <i>seq</i>  | Which command sequence need to reset, IP command, buffer command or both. |

### 23.5.24 static void QSPI\_EnableDDRMode ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | QSPI base pointer                                         |
| <i>enable</i> | True means enable DDR mode, false means disable DDR mode. |

### 23.5.25 void QSPI\_SetReadDataArea ( QuadSPI\_Type \* *base*, qspi\_read\_area\_t *area* )

This function can set the RX buffer readout, from AHB bus or IP Bus.

Parameters

|             |                                                               |
|-------------|---------------------------------------------------------------|
| <i>base</i> | QSPI base address.                                            |
| <i>area</i> | QSPI Rx buffer readout area. AHB bus buffer or IP bus buffer. |

### 23.5.26 void QSPI\_WriteBlocking ( QuadSPI\_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>   | QSPI base pointer                |
| <i>buffer</i> | The data bytes to send           |
| <i>size</i>   | The number of data bytes to send |

**23.5.27 static void QSPI\_WriteData ( QuadSPI\_Type \* *base*, uint32\_t *data* )  
[inline], [static]**

### Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | QSPI base pointer      |
| <i>data</i> | The data bytes to send |

**23.5.28 void QSPI\_ReadBlocking ( QuadSPI\_Type \* *base*, uint32\_t \* *buffer*, size\_t *size* )**

### Note

This function blocks via polling until all bytes have been sent. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | QSPI base pointer                   |
| <i>buffer</i> | The data bytes to send              |
| <i>size</i>   | The number of data bytes to receive |

**23.5.29 uint32\_t QSPI\_ReadData ( QuadSPI\_Type \* *base* )**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | QSPI base pointer |
|-------------|-------------------|

Returns

The data in the FIFO.

### 23.5.30 static void QSPI\_TransferSendBlocking ( QuadSPI\_Type \* *base*, qspi\_transfer\_t \* *xfer* ) [inline], [static]

This function writes a continuous data to the QSPI transmit FIFO. This function is a block function and can return only when finished. This function uses polling methods.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
| <i>xfer</i> | QSPI transfer structure. |

### 23.5.31 static void QSPI\_TransferReceiveBlocking ( QuadSPI\_Type \* *base*, qspi\_transfer\_t \* *xfer* ) [inline], [static]

This function reads continuous data from the QSPI receive buffer/FIFO. This function is a blocking function and can return only when finished. This function uses polling methods. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | Pointer to QuadSPI Type. |
| <i>xfer</i> | QSPI transfer structure. |

### 23.6 QSPI eDMA Driver

#### 23.6.1 Overview

### Data Structures

- struct `qspi_edma_handle_t`

*QSPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

### TypeDefs

- typedef void(\* `qspi_edma_callback_t`)(QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, status\_t status, void \*userData)

*QSPI eDMA transfer callback function for finish and error.*

### eDMA Transactional

- void `QSPI_TransferTxCreateHandleEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_edma_callback_t` callback, void \*userData, edma\_handle\_t \*dmaHandle)  
*Initializes the QSPI handle for send which is used in transactional functions and set the callback.*
- void `QSPI_TransferRxCreateHandleEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_edma_callback_t` callback, void \*userData, edma\_handle\_t \*dmaHandle)  
*Initializes the QSPI handle for receive which is used in transactional functions and set the callback.*
- status\_t `QSPI_TransferSendEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_transfer_t` \*xfer)  
*Transfers QSPI data using an eDMA non-blocking method.*
- status\_t `QSPI_TransferReceiveEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, `qspi_transfer_t` \*xfer)  
*Receives data using an eDMA non-blocking method.*
- void `QSPI_TransferAbortSendEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void `QSPI_TransferAbortReceiveEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle)  
*Aborts the receive data using eDMA.*
- status\_t `QSPI_TransferGetSendCountEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the transferred counts of send.*
- status\_t `QSPI_TransferGetReceiveCountEDMA` (QuadSPI\_Type \*base, qspi\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the status of the receive transfer.*

## 23.6.2 Data Structure Documentation

### 23.6.2.1 struct \_qspi\_edma\_handle

#### Data Fields

- `edma_handle_t * dmaHandle`  
*eDMA handler for QSPI send*
- `size_t transferSize`  
*Bytes need to transfer.*
- `uint8_t 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 QSPI eDMA transfer.*
- `qspi_edma_callback_t callback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*User callback parameter.*

#### 23.6.2.1.0.53 Field Documentation

##### 23.6.2.1.0.53.1 size\_t qspi\_edma\_handle\_t::transferSize

##### 23.6.2.1.0.53.2 uint8\_t qspi\_edma\_handle\_t::nbytes

## 23.6.3 Function Documentation

### 23.6.3.1 void QSPI\_TransferTxCreateHandleEDMA ( `QuadSPI_Type * base,` `qspi_edma_handle_t * handle, qspi_edma_callback_t callback, void * userData,` `edma_handle_t * dmaHandle )`

Parameters

|                          |                                                      |
|--------------------------|------------------------------------------------------|
| <code>base</code>        | QSPI peripheral base address                         |
| <code>handle</code>      | Pointer to <code>qspi_edma_handle_t</code> structure |
| <code>callback</code>    | QSPI callback, NULL means no callback.               |
| <code>userData</code>    | User callback function data.                         |
| <code>rxDmaHandle</code> | User requested eDMA handle for eDMA transfer         |

### 23.6.3.2 void QSPI\_TransferRxCreateHandleEDMA ( `QuadSPI_Type * base,` `qspi_edma_handle_t * handle, qspi_edma_callback_t callback, void * userData,` `edma_handle_t * dmaHandle )`

## QSPI eDMA Driver

Parameters

|                    |                                              |
|--------------------|----------------------------------------------|
| <i>base</i>        | QSPI peripheral base address                 |
| <i>handle</i>      | Pointer to qspi_edma_handle_t structure      |
| <i>callback</i>    | QSPI callback, NULL means no callback.       |
| <i>userData</i>    | User callback function data.                 |
| <i>rxDmaHandle</i> | User requested eDMA handle for eDMA transfer |

### 23.6.3.3 status\_t QSPI\_TransferSendEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle*, qspi\_transfer\_t \* *xfer* )

This function writes data to the QSPI transmit FIFO. This function is non-blocking.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                |
| <i>handle</i> | Pointer to qspi_edma_handle_t structure |
| <i>xfer</i>   | QSPI transfer structure.                |

### 23.6.3.4 status\_t QSPI\_TransferReceiveEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle*, qspi\_transfer\_t \* *xfer* )

This function receive data from the QSPI receive buffer/FIFO. This function is non-blocking. Users shall notice that this receive size shall not bigger than 64 bytes. As this interface is used to read flash status registers. For flash contents read, please use AHB bus read, this is much more efficiency.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                |
| <i>handle</i> | Pointer to qspi_edma_handle_t structure |
| <i>xfer</i>   | QSPI transfer structure.                |

### 23.6.3.5 void QSPI\_TransferAbortSendEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | QSPI peripheral base address.           |
| <i>handle</i> | Pointer to qspi_edma_handle_t structure |

### 23.6.3.6 void QSPI\_TransferAbortReceiveEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle* )

This function abort receive data which using eDMA.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | QSPI peripheral base address.           |
| <i>handle</i> | Pointer to qspi_edma_handle_t structure |

### 23.6.3.7 status\_t QSPI\_TransferGetSendCountEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                 |
| <i>handle</i> | Pointer to qspi_edma_handle_t structure. |
| <i>count</i>  | Bytes sent.                              |

Return values

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

### 23.6.3.8 status\_t QSPI\_TransferGetReceiveCountEDMA ( QuadSPI\_Type \* *base*, qspi\_edma\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

## QSPI eDMA Driver

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                |
| <i>handle</i> | Pointer to qspi_edma_handle_t structure |
| <i>count</i>  | Bytes received.                         |

Return values

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

# Chapter 24

## SAI: Serial Audio Interface

### 24.1 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.

### 24.2 Typical use case

#### 24.2.1 SAI 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 SAI_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_SAI_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...
```

## Typical use case

```
SAI_TxGetDefaultConfig(&user_config);

SAI_TxInit(SAI0, &user_config);
SAI_TransferTxCreateHandle(SAI0, &g_saiHandle, SAI_UserCallback, NULL);

//Configure sai format
SAI_TransferTxSetTransferFormat(SAI0, &g_saiHandle, mclkSource, mclk);

// Prepare to send.
sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Send out.
SAI_TransferSendNonBlocking(SAI0, &g_saiHandle, &sendXfer);

// Wait send finished.
while (!txFinished)
{
}

// ...
}
```

### 24.2.2 SAI 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 SAI_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_SAI_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    SAI_TxGetDefaultConfig(&user_config);
    SAI_TxInit(SAI0, &user_config);

    // Sets up the DMA.
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, SAI_TX_DMA_CHANNEL, SAI_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, SAI_TX_DMA_CHANNEL);

    DMA_Init(DMA0);

    /* Creates the DMA handle. */
    DMA_CreateHandle(&g_saiTxDmaHandle, DMA0, SAI_TX_DMA_CHANNEL);

    SAI_TransferTxCreateHandleDMA(SAI0, &g_saiTxDmaHandle, SAI_UserCallback,
        NULL);

    // Prepares to send.
```

```

sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData) / sizeof(sendData[0]);
txFinished = false;

// Sends out.
SAI_TransferSendDMA(&g_saiHandle, &sendXfer);

// Waits for send to complete.
while (!txFinished)
{
}

// ...
}

```

## Modules

- SAI DMA Driver
- SAI SDMA Driver
- SAI eDMA Driver

## 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) }

## Typical use case

- 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_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` }  
*Mater 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` {
   
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_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, 2))

*Version 2.1.2.*

## 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 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)

## Typical use case

- 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.

## Interrupts

- static void **SAI\_TxEnableInterrupts** (I2S\_Type \*base, uint32\_t mask)  
Enables the SAI Tx interrupt requests.
- static void **SAI\_RxEnableInterrupts** (I2S\_Type \*base, uint32\_t mask)  
Enables the SAI Rx interrupt requests.
- static void **SAI\_TxDisableInterrupts** (I2S\_Type \*base, uint32\_t mask)  
Disables the SAI Tx interrupt requests.
- static void **SAI\_RxDisableInterrupts** (I2S\_Type \*base, uint32\_t mask)  
Disables the SAI Rx interrupt requests.

## DMA Control

- static void **SAI\_TxEnableDMA** (I2S\_Type \*base, uint32\_t mask, bool enable)  
Enables/disables the SAI Tx DMA requests.
- static void **SAI\_RxEnableDMA** (I2S\_Type \*base, uint32\_t mask, bool enable)  
Enables/disables the SAI Rx DMA requests.
- static uint32\_t **SAI\_TxGetDataRegisterAddress** (I2S\_Type \*base, uint32\_t channel)  
Gets the SAI Tx data register address.
- static uint32\_t **SAI\_RxGetDataRegisterAddress** (I2S\_Type \*base, uint32\_t channel)  
Gets the SAI Rx data register address.

## Bus Operations

- void **SAI\_TxSetFormat** (I2S\_Type \*base, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void **SAI\_RxSetFormat** (I2S\_Type \*base, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- void **SAI\_WriteBlocking** (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Sends data using a blocking method.*
- 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 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)

## Data Structure Documentation

- 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.*

### 24.3 Data Structure Documentation

#### 24.3.1 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.*

#### 24.3.2 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.*
- [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.*

**24.3.2.0.0.54 Field Documentation****24.3.2.0.0.54.1 uint8\_t sai\_transfer\_format\_t::channel****24.3.3 struct sai\_transfer\_t****Data Fields**

- `uint8_t * data`  
*Data start address to transfer.*
- `size_t dataSize`  
*Transfer size.*

**24.3.3.0.0.55 Field Documentation****24.3.3.0.0.55.1 uint8\_t\* sai\_transfer\_t::data****24.3.3.0.0.55.2 size\_t sai\_transfer\_t::dataSize****24.3.4 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.*

**24.4 Macro Definition Documentation****24.4.1 #define SAI\_XFER\_QUEUE\_SIZE (4)**

## Enumeration Type Documentation

### 24.5 Enumeration Type Documentation

#### 24.5.1 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.

#### 24.5.2 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.

#### 24.5.3 enum sai\_master\_slave\_t

Enumerator

- kSAI\_Master* Master mode.
- kSAI\_Slave* Slave mode.

#### 24.5.4 enum sai\_mono\_stereo\_t

Enumerator

- kSAI\_Stereo* Stereo sound.
- kSAI\_MonoRight* Only Right channel have sound.
- kSAI\_MonoLeft* Only left channel have sound.

## 24.5.5 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.

## 24.5.6 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.

## 24.5.7 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.

## 24.5.8 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.

## 24.5.9 enum \_sai\_dma\_enable\_t

Enumerator

*kSAI\_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.

## Enumeration Type Documentation

*kSAI\_FIFORequestDMAEnable* FIFO request caused by the DMA request.

### 24.5.10 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.

### 24.5.11 enum sai\_reset\_type\_t

Enumerator

*kSAI\_ResetTypeSoftware* Software reset, reset the logic state.

*kSAI\_ResetTypeFIFO* FIFO reset, reset the FIFO read and write pointer.

*kSAI\_ResetAll* All reset.

### 24.5.12 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.

### 24.5.13 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.

## 24.6 Function Documentation

### 24.6.1 void SAI\_TxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )

Ungates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_TxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

### 24.6.2 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. |

### 24.6.3 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);
```

## Function Documentation

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | pointer to master configuration structure |
|---------------|-------------------------------------------|

### 24.6.4 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 |
|---------------|-------------------------------------------|

### 24.6.5 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 |
|-------------|------------------|

### 24.6.6 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 |
|-------------|------------------|

### 24.6.7 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 |
|-------------|------------------|

#### 24.6.8 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. |

#### 24.6.9 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. |

#### 24.6.10 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.

#### 24.6.11 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> |

**24.6.12 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.

**24.6.13 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> |

**24.6.14 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 |

#### 24.6.15 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 |

#### 24.6.16 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. |

#### 24.6.17 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. |

#### 24.6.18 static void SAI\_TxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

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> |

**24.6.19 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> |

**24.6.20 static void SAI\_TxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                          |
| <i>mask</i> | <p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**24.6.21 static void SAI\_RxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                          |
| <i>mask</i> | <p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**24.6.22 static void SAI\_TxEnableDMA ( I2S\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

Parameters

---

## Function Documentation

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

**24.6.23 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.                                                                                                                                                    |

**24.6.24 static uint32\_t SAI\_TxGetDataRegisterAddress ( I2S\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | SAI base pointer.        |
| <i>channel</i> | Which data channel used. |

Returns

data register address.

**24.6.25 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.

#### 24.6.26 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. |

#### 24.6.27 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. |

## Function Documentation

**24.6.28 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.                                     |

**24.6.29 static void SAI\_WriteData ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *data* ) [inline], [static]**

Parameters

|                |                           |
|----------------|---------------------------|
| <i>base</i>    | SAI base pointer.         |
| <i>channel</i> | Data channel used.        |
| <i>data</i>    | Data needs to be written. |

**24.6.30 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.                                        |

24.6.31 **static uint32\_t SAI\_ReadData ( I2S\_Type \* *base*, uint32\_t *channel* )**  
[**inline**], [**static**]

## Function Documentation

Parameters

|                |                    |
|----------------|--------------------|
| <i>base</i>    | SAI base pointer.  |
| <i>channel</i> | Data channel used. |

Returns

Data in SAI FIFO.

### **24.6.32 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 |

### **24.6.33 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. |

24.6.34 **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.

## Function Documentation

Parameters

|                           |                                                                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                   |
| <i>handle</i>             | SAI handle pointer.                                                                                                                 |
| <i>format</i>             | Pointer to the SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                            |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format. |

Returns

Status of this function. Return value is the status\_t.

**24.6.35 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.

**24.6.36 status\_t SAI\_TransferSendNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

This API returns immediately after the transfer initiates. Call the SAI\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <a href="#">sai_transfer_t</a> 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.        |

#### 24.6.37 **status\_t SAI\_TransferReceiveNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

This API returns immediately after the transfer initiates. Call the SAI\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer                                                       |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <a href="#">sai_transfer_t</a> 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.        |

## Function Documentation

24.6.38 **status\_t SAI\_TransferGetSendCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count sent.                                                      |

Return values

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

#### 24.6.39 **status\_t SAI\_TransferGetReceiveCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count received.                                                  |

Return values

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

#### 24.6.40 **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.

## Function Documentation

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |

### **24.6.41 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. |

### **24.6.42 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. |

### **24.6.43 void SAI\_TransferTerminateReceive ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )**

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortReceive.

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

**24.6.44 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. |

**24.6.45 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. |

### 24.7 SAI DMA Driver

#### 24.7.1 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.*

#### 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)  
*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.*

## 24.7.2 Data Structure Documentation

### 24.7.2.1 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.*

#### 24.7.2.1.0.56 Field Documentation

##### 24.7.2.1.0.56.1 `sai_transfer_t sai_dma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`

##### 24.7.2.1.0.56.2 `volatile uint8_t sai_dma_handle_t::queueUser`

## 24.7.3 Function Documentation

### 24.7.3.1 `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

## SAI DMA Driver

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

**24.7.3.2 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. |

**24.7.3.3 void SAI\_TransferTxSetFormatDMA ( 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 the eDMA parameter according to the 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. |

#### 24.7.3.4 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. |

#### 24.7.3.5 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.

## SAI DMA Driver

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.      |

### 24.7.3.6 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.      |

### 24.7.3.7 void SAI\_TransferAbortSendDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle* )

Parameters

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

#### 24.7.3.8 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. |

#### 24.7.3.9 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. |

#### 24.7.3.10 status\_t SAI\_TransferGetReceiveCountDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|

## SAI DMA Driver

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

## 24.8 SAI eDMA Driver

### 24.8.1 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.*

#### 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.*

## SAI eDMA Driver

- status\_t **SAI\_TransferGetReceiveCountEDMA** (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count received by SAI.*

### 24.8.2 Data Structure Documentation

#### 24.8.2.1 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.*
- edma\_tcd\_t **tdc [SAI\_XFER\_QUEUE\_SIZE+1U]**  
*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.*

#### 24.8.2.1.0.57 Field Documentation

24.8.2.1.0.57.1 `uint8_t sai_edma_handle_t::nbytes`

24.8.2.1.0.57.2 `edma_tcd_t sai_edma_handle_t::tcd[SAI_XFER_QUEUE_SIZE+1U]`

24.8.2.1.0.57.3 `sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`

24.8.2.1.0.57.4 `volatile uint8_t sai_edma_handle_t::queueUser`

#### 24.8.3 Function Documentation

24.8.3.1 `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.

## SAI eDMA Driver

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. |

**24.8.3.2 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. |

**24.8.3.3 void SAI\_TransferTxSetFormatEDMA ( 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>mclkSourceClockHz</i> | SAI master clock source frequency in Hz.                                                                                        |
| <i>bclkSourceClockHz</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. |

#### 24.8.3.4 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>mclkSourceClockHz</i> | SAI master clock source frequency in Hz.                                                                                             |
| <i>bclkSourceClockHz</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. |

#### 24.8.3.5 status\_t SAI\_TransferSendEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## SAI eDMA Driver

### 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.           |

### 24.8.3.6 **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.            |

### 24.8.3.7 **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.

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

#### 24.8.3.8 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. |

#### 24.8.3.9 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. |

#### 24.8.3.10 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 |
|-------------|------------------|

## SAI eDMA Driver

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | SAI eDMA handle pointer. |
|---------------|--------------------------|

### 24.8.3.11 **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. |

### 24.8.3.12 **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. |

## 24.9 SAI SDMA Driver

### 24.9.1 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.*

#### 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.*

### 24.9.2 Data Structure Documentation

#### 24.9.2.1 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.*

#### 24.9.2.1.0.58 Field Documentation

24.9.2.1.0.58.1 `sdma_buffer_descriptor_t sai_sdma_handle_t::bdPool[SAI_XFER_QUEUE_SIZE]`

24.9.2.1.0.58.2 `sai_transfer_t sai_sdma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`

24.9.2.1.0.58.3 `volatile uint8_t sai_sdma_handle_t::queueUser`

#### 24.9.3 Function Documentation

24.9.3.1 `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.

## SAI SDMA Driver

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. |

**24.9.3.2 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. |

**24.9.3.3 void SAI\_TransferTxSetFormatSDMA ( 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>mclkSourceClockHz</i> | SAI master clock source frequency in Hz.                                                                                        |
| <i>bclkSourceClockHz</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. |

**24.9.3.4 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>mclkSourceClockHz</i> | SAI master clock source frequency in Hz.                                                                                             |
| <i>bclkSourceClockHz</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. |

**24.9.3.5 status\_t SAI\_TransferSendSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## SAI SDMA Driver

### 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.           |

### 24.9.3.6 **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\_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 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.            |

### 24.9.3.7 **void SAI\_TransferAbortSendSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle* )**

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI SDMA handle pointer. |

#### 24.9.3.8 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 25

# SDMA: Smart Direct Memory Access (SDMA) Controller Driver

## 25.1 Overview

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

## 25.2 Typical use case

### 25.2.1 SDMA Operation

```
sdma_transfer_config_t transferConfig;
sdma_config_t userConfig;
uint32_t transferDone = false;

SDMA_GetDefaultConfig(&userConfig);
SDMA_Init(SDMAARM, &userConfig);
SDMA_CreateHandle(&g_SDMA_Handle, SDMAARM, channel);
SDMA_SetCallback(&g_SDMA_Handle, SDMA_Callback, &transferDone);
SDMA_PreparesTransfer(&transferConfig, srcAddr, srcWidth, destAddr, destWidth,
                      bytesEachRequest, transferBytes, kSDMA_PeripheralTypeMemory,
                      kSDMA_MemoryToMemory);
SDMA_SetChannelPriority(SDMAARM, channel, 2U);
SDMA_SubmitTransfer(&g_SDMA_Handle, &transferConfig);
SDMA_StartTransfer(&g_SDMA_Handle);
/* Waits for the SDMA transfer to finish */
while (transferDone != true);
```

## Data Structures

- struct [sdma\\_config\\_t](#)  
*SDMA global configuration structure.* [More...](#)
- struct [sdma\\_transfer\\_config\\_t](#)  
*SDMA transfer configuration.* [More...](#)
- struct [sdma\\_buffer\\_descriptor\\_t](#)  
*SDMA buffer descriptor structure.* [More...](#)
- struct [sdma\\_channel\\_control\\_t](#)  
*SDMA channel control descriptor structure.* [More...](#)
- struct [sdma\\_context\\_data\\_t](#)  
*SDMA context structure for each channel.* [More...](#)
- struct [sdma\\_handle\\_t](#)  
*SDMA transfer handle structure.* [More...](#)

## Typedefs

- typedef void(\* [sdma\\_callback](#)) (struct \_sdma\_handle \*handle, void \*userData, bool transferDone, uint32\_t bdIndex)  
*Define callback function for SDMA.*

## Typical use case

## Enumerations

- enum `sdma_transfer_size_t` {  
    `kSDMA_TransferSize1Bytes` = 0x1U,  
    `kSDMA_TransferSize2Bytes` = 0x2U,  
    `kSDMA_TransferSize4Bytes` = 0x0U }  
    *SDMA transfer configuration.*
- enum `sdma_bd_status_t` {  
    `kSDMA_BDStatusDone` = 0x1U,  
    `kSDMA_BDStatusWrap` = 0x2U,  
    `kSDMA_BDStatusContinuous` = 0x4U,  
    `kSDMA_BDStatusInterrupt` = 0x8U,  
    `kSDMA_BDStatusError` = 0x10U,  
    `kSDMA_BDStatusLast`,  
    `kSDMA_BDStatusExtend` = 0x80 }  
    *SDMA buffer descriptor status.*
- enum `sdma_bd_command_t` {  
    `kSDMA_BDCommandSETDM` = 0x1U,  
    `kSDMA_BDCommandGETDM` = 0x2U,  
    `kSDMA_BDCommandSETPM` = 0x4U,  
    `kSDMA_BDCommandGETPM` = 0x6U,  
    `kSDMA_BDCommandSETCTX` = 0x7U,  
    `kSDMA_BDCommandGETCTX` = 0x3U }  
    *SDMA buffer descriptor command.*
- enum `sdma_context_switch_mode_t` {  
    `kSDMA_ContextSwitchModeStatic` = 0x0U,  
    `kSDMA_ContextSwitchModeDynamicLowPower`,  
    `kSDMA_ContextSwitchModeDynamicWithNoLoop`,  
    `kSDMA_ContextSwitchModeDynamic` }  
    *SDMA context switch mode.*
- enum `sdma_clock_ratio_t` {  
    `kSDMA_HalfARMClockFreq` = 0x0U,  
    `kSDMA_ARMClockFreq` }  
    *SDMA core clock frequency ratio to the ARM DMA interface.*
- enum `sdma_transfer_type_t` {  
    `kSDMA_MemoryToMemory` = 0x0U,  
    `kSDMA_PeripheralToMemory`,  
    `kSDMA_MemoryToPeripheral` }  
    *SDMA transfer type.*
- enum `sdma_peripheral_t` {  
    `kSDMA_PeripheralTypeMemory` = 0x0,  
    `kSDMA_PeripheralTypeUART`,  
    `kSDMA_PeripheralTypeUART_SP`,  
    `kSDMA_PeripheralTypeSPDIF`,  
    `kSDMA_PeripheralNormal`,  
    `kSDMA_PeripheralNormal_SP` }  
    *Peripheral type use SDMA.*

- enum `_sdma_transfer_status` {
   
    `kStatus_SDMA_ERROR` = MAKE\_STATUS(kStatusGroup\_SDMA, 0),
   
    `kStatus_SDMA_Busy` = MAKE\_STATUS(kStatusGroup\_SDMA, 1) }
   
*SDMA transfer status.*

## Driver version

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

## SDMA initialization and de-initialization

- void `SDMA_Init` (SDMAARM\_Type \*base, const `sdma_config_t` \*config)
   
*Initializes the SDMA peripheral.*
- void `SDMA_Deinit` (SDMAARM\_Type \*base)
   
*Deinitializes the SDMA peripheral.*
- void `SDMA_GetDefaultConfig` (`sdma_config_t` \*config)
   
*Gets the SDMA default configuration structure.*
- void `SDMA_ResetModule` (SDMAARM\_Type \*base)
   
*Sets all SDMA core register to reset status.*

## SDMA Channel Operation

- static void `SDMA_EnableChannelErrorInterrupts` (SDMAARM\_Type \*base, uint32\_t channel)
   
*Enables the interrupt source for the SDMA error.*
- static void `SDMA_DisableChannelErrorInterrupts` (SDMAARM\_Type \*base, uint32\_t channel)
   
*Disables the interrupt source for the SDMA error.*

## SDMA Buffer Descriptor Operation

- void `SDMA_ConfigBufferDescriptor` (`sdma_buffer_descriptor_t` \*bd, uint32\_t srcAddr, uint32\_t destAddr, `sdma_transfer_size_t` busWidth, size\_t bufferSize, bool isLast, bool enableInterrupt, bool isWrap)
   
*Sets buffer descriptor contents.*

## SDMA Channel Transfer Operation

- static void `SDMA_SetChannelPriority` (SDMAARM\_Type \*base, uint32\_t channel, uint8\_t priority)
   
*Set SDMA channel priority.*
- static void `SDMA_SetSourceChannel` (SDMAARM\_Type \*base, uint32\_t source, uint32\_t channelMask)
   
*Set SDMA request source mapping channel.*
- static void `SDMA_StartChannelSoftware` (SDMAARM\_Type \*base, uint32\_t channel)
   
*Start a SDMA channel by software trigger.*
- static void `SDMA_StartChannelEvents` (SDMAARM\_Type \*base, uint32\_t channel)
   
*Start a SDMA channel by hardware events.*
- static void `SDMA_StopChannel` (SDMAARM\_Type \*base, uint32\_t channel)
   
*Stop a SDMA channel.*

## Typical use case

- void **SDMA\_SetContextSwitchMode** (SDMAARM\_Type \*base, **sdma\_context\_switch\_mode\_t** mode)  
*Set the SDMA context switch mode.*

## SDMA Channel Status Operation

- static uint32\_t **SDMA\_GetChannelInterruptStatus** (SDMAARM\_Type \*base)  
*Gets the SDMA interrupt status of all channels.*
- static void **SDMA\_ClearChannelInterruptStatus** (SDMAARM\_Type \*base, uint32\_t mask)  
*Clear the SDMA channel interrupt status of specific channels.*
- static uint32\_t **SDMA\_GetChannelStopStatus** (SDMAARM\_Type \*base)  
*Gets the SDMA stop status of all channels.*
- static void **SDMA\_ClearChannelStopStatus** (SDMAARM\_Type \*base, uint32\_t mask)  
*Clear the SDMA channel stop status of specific channels.*
- static uint32\_t **SDMA\_GetChannelPendingStatus** (SDMAARM\_Type \*base)  
*Gets the SDMA channel pending status of all channels.*
- static void **SDMA\_ClearChannelPendingStatus** (SDMAARM\_Type \*base, uint32\_t mask)  
*Clear the SDMA channel pending status of specific channels.*
- static uint32\_t **SDMA\_GetErrorStatus** (SDMAARM\_Type \*base)  
*Gets the SDMA channel error status.*
- bool **SDMA\_GetRequestSourceStatus** (SDMAARM\_Type \*base, uint32\_t source)  
*Gets the SDMA request source pending status.*

## SDMA Transactional Operation

- void **SDMA\_CreateHandle** (sdma\_handle\_t \*handle, SDMAARM\_Type \*base, uint32\_t channel, **sdma\_context\_data\_t** \*context)  
*Creates the SDMA handle.*
- void **SDMA\_InstallBDMemory** (sdma\_handle\_t \*handle, **sdma\_buffer\_descriptor\_t** \*BDPool, uint32\_t BDCount)  
*Installs the BDs memory pool into the SDMA handle.*
- void **SDMA\_SetCallback** (sdma\_handle\_t \*handle, **sdma\_callback** callback, void \*userData)  
*Installs a callback function for the SDMA transfer.*
- void **SDMA\_PrepareTransfer** (**sdma\_transfer\_config\_t** \*config, uint32\_t srcAddr, uint32\_t destAddr, uint32\_t srcWidth, uint32\_t destWidth, uint32\_t bytesEachRequest, uint32\_t transferSize, uint32\_t eventSource, **sdma\_peripheral\_t** peripheral, **sdma\_transfer\_type\_t** type)  
*Prepares the SDMA transfer structure.*
- void **SDMA\_SubmitTransfer** (sdma\_handle\_t \*handle, const **sdma\_transfer\_config\_t** \*config)  
*Submits the SDMA transfer request.*
- void **SDMA\_StartTransfer** (sdma\_handle\_t \*handle)  
*SDMA starts transfer.*
- void **SDMA\_StopTransfer** (sdma\_handle\_t \*handle)  
*SDMA stops transfer.*
- void **SDMA\_AbortTransfer** (sdma\_handle\_t \*handle)  
*SDMA aborts transfer.*
- void **SDMA\_HandleIRQ** (sdma\_handle\_t \*handle)  
*SDMA IRQ handler for complete a buffer descriptor transfer.*

## 25.3 Data Structure Documentation

### 25.3.1 struct sdma\_config\_t

#### Data Fields

- bool `enableRealTimeDebugPin`  
*If enable real-time debug pin, default is closed to reduce power consumption.*
- bool `isSoftwareResetClearLock`  
*If software reset clears the LOCK bit which prevent writing SDMA scripts into SDMA.*
- `sdma_clock_ratio_t ratio`  
*SDMA core clock ratio to ARM platform DMA interface.*

#### 25.3.1.0.0.59 Field Documentation

##### 25.3.1.0.0.59.1 bool sdma\_config\_t::enableRealTimeDebugPin

##### 25.3.1.0.0.59.2 bool sdma\_config\_t::isSoftwareResetClearLock

### 25.3.2 struct sdma\_transfer\_config\_t

This structure configures the source/destination transfer attribute.

#### Data Fields

- `uint32_t srcAddr`  
*Source address of the transfer.*
- `uint32_t destAddr`  
*Destination address of the transfer.*
- `sdma_transfer_size_t srcTransferSize`  
*Source data transfer size.*
- `sdma_transfer_size_t destTransferSize`  
*Destination data transfer size.*
- `uint32_t bytesPerRequest`  
*Bytes to transfer in a minor loop.*
- `uint32_t transferSzie`  
*Bytes to transfer for this descriptor.*
- `uint32_t scriptAddr`  
*SDMA script address located in SDMA ROM.*
- `uint32_t eventSource`  
*Event source number for the channel.*
- bool `isEventIgnore`  
*True means software trigger, false means hardware trigger.*
- bool `isSoftTriggerIgnore`  
*If ignore the HE bit, 1 means use hardware events trigger, 0 means software trigger.*
- `sdma_transfer_type_t type`  
*Transfer type, transfer type used to decide the SDMA script.*

## Data Structure Documentation

### 25.3.2.0.0.60 Field Documentation

25.3.2.0.0.60.1 `sdma_transfer_size_t sdma_transfer_config_t::srcTransferSize`

25.3.2.0.0.60.2 `sdma_transfer_size_t sdma_transfer_config_t::destTransferSize`

25.3.2.0.0.60.3 `uint32_t sdma_transfer_config_t::scriptAddr`

25.3.2.0.0.60.4 `uint32_t sdma_transfer_config_t::eventSource`

0 means no event, use software trigger

25.3.2.0.0.60.5 `sdma_transfer_type_t sdma_transfer_config_t::type`

### 25.3.3 `struct sdma_buffer_descriptor_t`

This structure is a buffer descriptor, this structure describes the buffer start address and other options

#### Data Fields

- `uint32_t count`: 16  
*Bytes of the buffer length for this buffer descriptor.*
- `uint32_t status`: 8  
*E,R,I,C,W,D status bits stored here.*
- `uint32_t command`: 8  
*command mostly used for channel 0*
- `uint32_t bufferAddr`  
*Buffer start address for this descriptor.*
- `uint32_t extendBufferAddr`  
*External buffer start address, this is an optional for a transfer.*

### 25.3.3.0.0.61 Field Documentation

25.3.3.0.0.61.1 `uint32_t sdma_buffer_descriptor_t::count`

25.3.3.0.0.61.2 `uint32_t sdma_buffer_descriptor_t::bufferAddr`

25.3.3.0.0.61.3 `uint32_t sdma_buffer_descriptor_t::extendBufferAddr`

### 25.3.4 `struct sdma_channel_control_t`

#### Data Fields

- `uint32_t currentBDAddr`  
*Address of current buffer descriptor processed.*
- `uint32_t baseBDAddr`  
*The start address of the buffer descriptor array.*
- `uint32_t channelDesc`

- *Optional for transfer.*
- `uint32_t status`  
*Channel status.*

### 25.3.5 struct sdma\_context\_data\_t

This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

#### Data Fields

- `uint32_t GeneralReg [8]`  
*8 general registers used for SDMA RISC core*

### 25.3.6 struct sdma\_handle\_t

#### Data Fields

- `sdma_callback callback`  
*Callback function for major count exhausted.*
- `void *userData`  
*Callback function parameter.*
- `SDMAARM_Type *base`  
*SDMA peripheral base address.*
- `sdma_buffer_descriptor_t *BDPool`  
*Pointer to memory stored BD arrays.*
- `uint32_t bdCount`  
*How many buffer descriptor.*
- `uint32_t bdIndex`  
*How many buffer descriptor.*
- `uint32_t eventSource`  
*Event source count for the channel.*
- `sdma_context_data_t *context`  
*Channel context to execute in SDMA.*
- `uint8_t channel`  
*SDMA channel number.*
- `uint8_t flags`  
*The status of the current channel.*

## Enumeration Type Documentation

### 25.3.6.0.0.62 Field Documentation

25.3.6.0.0.62.1 `sdma_callback` `sdma_handle_t::callback`

25.3.6.0.0.62.2 `void*` `sdma_handle_t::userData`

25.3.6.0.0.62.3 `SDMAARM_Type*` `sdma_handle_t::base`

25.3.6.0.0.62.4 `sdma_buffer_descriptor_t*` `sdma_handle_t::BDPool`

25.3.6.0.0.62.5 `uint8_t` `sdma_handle_t::channel`

25.3.6.0.0.62.6 `uint8_t` `sdma_handle_t::flags`

## 25.4 Macro Definition Documentation

25.4.1 `#define FSL_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

Version 2.0.0.

## 25.5 Typedef Documentation

25.5.1 `typedef void(* sdma_callback)(struct _sdma_handle *handle, void *userData, bool transferDone, uint32_t bdIndex)`

## 25.6 Enumeration Type Documentation

### 25.6.1 enum `sdma_transfer_size_t`

Enumerator

`kSDMA_TransferSize1Bytes` Source/Destination data transfer size is 1 byte every time.

`kSDMA_TransferSize2Bytes` Source/Destination data transfer size is 2 bytes every time.

`kSDMA_TransferSize4Bytes` Source/Destination data transfer size is 4 bytes every time.

### 25.6.2 enum `sdma_bd_status_t`

Enumerator

`kSDMA_BDStatusDone` BD ownership, 0 means ARM core owns the BD, while 1 means SDMA owns BD.

`kSDMA_BDStatusWrap` While this BD is last one, the next BD will be the first one.

`kSDMA_BDStatusContinuous` Buffer is allowed to transfer/receive to/from multiple buffers.

`kSDMA_BDStatusInterrupt` While this BD finished, send an interrupt.

`kSDMA_BDStatusError` Error occurred on buffer descriptor command.

`kSDMA_BDStatusLast` This BD is the last BD in this array. It means the transfer ended after this buffer

***kSDMA\_BDStatusExtend*** Buffer descriptor extend status for SDMA scripts.

### 25.6.3 enum sdma\_bd\_command\_t

Enumerator

***kSDMA\_BDCommandSETDM*** Load SDMA data memory from ARM core memory buffer.

***kSDMA\_BDCommandGETDM*** Copy SDMA data memory to ARM core memory buffer.

***kSDMA\_BDCommandSETPM*** Load SDMA program memory from ARM core memory buffer.

***kSDMA\_BDCommandGETPM*** Copy SDMA program memory to ARM core memory buffer.

***kSDMA\_BDCommandSETCTX*** Load context for one channel into SDMA RAM from ARM platform memory buffer.

***kSDMA\_BDCommandGETCTX*** Copy context for one channel from SDMA RAM to ARM platform memory buffer.

### 25.6.4 enum sdma\_context\_switch\_mode\_t

Enumerator

***kSDMA\_ContextSwitchModeStatic*** SDMA context switch mode static.

***kSDMA\_ContextSwitchModeDynamicLowPower*** SDMA context switch mode dynamic with low power.

***kSDMA\_ContextSwitchModeDynamicWithNoLoop*** SDMA context switch mode dynamic with no loop.

***kSDMA\_ContextSwitchModeDynamic*** SDMA context switch mode dynamic.

### 25.6.5 enum sdma\_clock\_ratio\_t

Enumerator

***kSDMA\_HalfARMClockFreq*** SDMA core clock frequency half of ARM platform.

***kSDMA\_ARMClockFreq*** SDMA core clock frequency equals to ARM platform.

### 25.6.6 enum sdma\_transfer\_type\_t

Enumerator

***kSDMA\_MemoryToMemory*** Transfer from memory to memory.

***kSDMA\_PeripheralToMemory*** Transfer from peripheral to memory.

***kSDMA\_MemoryToPeripheral*** Transfer from memory to peripheral.

## Function Documentation

### 25.6.7 enum sdma\_peripheral\_t

Enumerator

*kSDMA\_PeripheralTypeMemory* Peripheral DDR memory.

*kSDMA\_PeripheralTypeUART* UART use SDMA.

*kSDMA\_PeripheralTypeUART\_SP* UART instance in SPBA use SDMA.

*kSDMA\_PeripheralTypeSPDIF* SPDIF use SDMA.

*kSDMA\_PeripheralNormal* Normal peripheral use SDMA.

*kSDMA\_PeripheralNormal\_SP* Normal peripheral in SPBA use SDMA.

### 25.6.8 enum \_sdma\_transfer\_status

Enumerator

*kStatus\_SDMA\_ERROR* SDMA context error.

*kStatus\_SDMA\_Busy* Channel is busy and can't handle the transfer request.

## 25.7 Function Documentation

### 25.7.1 void SDMA\_Init ( **SDMAARM\_Type** \* *base*, **const sdma\_config\_t** \* *config* )

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

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>base</i>   | SDMA peripheral base address.                                  |
| <i>config</i> | A pointer to the configuration structure, see "sdma_config_t". |

Note

This function enables the minor loop map feature.

### 25.7.2 void SDMA\_Deinit ( **SDMAARM\_Type** \* *base* )

This function gates the SDMA clock.

Parameters

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

### 25.7.3 void SDMA\_GetDefaultConfig ( **sdma\_config\_t** \* *config* )

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

```
* config.enableRealTimeDebugPin = false;
* config.isSoftwareResetClearLock = true;
* config.ratio = kSDMA_HalfARMClockFreq;
*
```

Parameters

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

### 25.7.4 void SDMA\_ResetModule ( **SDMAARM\_Type** \* *base* )

If only reset ARM core, SDMA register cannot return to reset value, shall call this function to reset all SDMA register to reset value. But the internal status cannot be reset.

Parameters

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

### 25.7.5 static void SDMA\_EnableChannelErrorInterrupts ( **SDMAARM\_Type** \* *base*, **uint32\_t** *channel* ) [inline], [static]

Enable this will trigger an interrupt while SDMA occurs error while executing scripts.

Parameters

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

### 25.7.6 static void SDMA\_DisableChannelErrorInterrupts ( **SDMAARM\_Type** \* *base*, **uint32\_t** *channel* ) [inline], [static]

## Function Documentation

Parameters

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

**25.7.7 void SDMA\_ConfigBufferDescriptor ( *sdma\_buffer\_descriptor\_t \* bd,*  
*uint32\_t srcAddr, uint32\_t destAddr, sdma\_transfer\_size\_t busWidth, size\_t*  
*bufferSize, bool isLast, bool enableInterrupt, bool isWrap*  )**

This function sets the descriptor contents such as source, dest address and status bits.

Parameters

|                        |                                                                                                                                                    |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>bd</i>              | Pointer to the buffer descriptor structure.                                                                                                        |
| <i>srcAddr</i>         | Source address for the buffer descriptor.                                                                                                          |
| <i>destAddr</i>        | Destination address for the buffer descriptor.                                                                                                     |
| <i>busWidth</i>        | The transfer width, it only can be a member of <i>sdma_transfer_size_t</i> .                                                                       |
| <i>bufferSize</i>      | Buffer size for this descriptor, this number shall less than 0xFFFF. If need to transfer a big size, shall divide into several buffer descriptors. |
| <i>isLast</i>          | Is the buffer descriptor the last one for the channel to transfer. If only one descriptor used for the channel, this bit shall set to TRUE.        |
| <i>enableInterrupt</i> | If trigger an interrupt while this buffer descriptor transfer finished.                                                                            |
| <i>isWrap</i>          | Is the buffer descriptor need to be wrapped. While this bit set to true, it will automatically wrap to the first buffer descriptor to do transfer. |

**25.7.8 static void SDMA\_SetChannelPriority ( *SDMAARM\_Type \* base, uint32\_t*  
*channel, uint8\_t priority*  ) [inline], [static]**

This function sets the channel priority. The default value is 0 for all channels, priority 0 will prevents channel from starting, so the priority must be set before start a channel.

Parameters

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

|                 |                        |
|-----------------|------------------------|
| <i>channel</i>  | SDMA channel number.   |
| <i>priority</i> | SDMA channel priority. |

### 25.7.9 static void SDMA\_SetSourceChannel ( SDMAARM\_Type \* *base*, uint32\_t *source*, uint32\_t *channelMask* ) [inline], [static]

This function sets which channel will be triggered by the dma request source.

Parameters

|                    |                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | SDMA peripheral base address.                                                                                                                                                       |
| <i>source</i>      | SDMA dma request source number.                                                                                                                                                     |
| <i>channelMask</i> | SDMA channel mask. 1 means channel 0, 2 means channel 1, 4 means channel 3. SDMA supports an event trigger multi-channel. A channel can also be triggered by several source events. |

### 25.7.10 static void SDMA\_StartChannelSoftware ( SDMAARM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function start a channel.

Parameters

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

### 25.7.11 static void SDMA\_StartChannelEvents ( SDMAARM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function start a channel.

Parameters

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

## Function Documentation

**25.7.12 static void SDMA\_StopChannel ( SDMAARM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function stops a channel.

Parameters

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

**25.7.13 void SDMA\_SetContextSwitchMode ( SDMAARM\_Type \* *base*, sdma\_context\_switch\_mode\_t *mode* )**

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | SDMA peripheral base address. |
| <i>mode</i> | SDMA context switch mode.     |

**25.7.14 static uint32\_t SDMA\_GetChannelInterruptStatus ( SDMAARM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The interrupt status for all channels. Check the relevant bits for specific channel.

**25.7.15 static void SDMA\_ClearChannelInterruptStatus ( SDMAARM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | SDMA peripheral base address.            |
| <i>mask</i> | The interrupt status need to be cleared. |

**25.7.16 static uint32\_t SDMA\_GetChannelStopStatus ( SDMAARM\_Type \* *base* ) [inline], [static]**

## Function Documentation

Parameters

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

Returns

The stop status for all channels. Check the relevant bits for specific channel.

### 25.7.17 static void SDMA\_ClearChannelStopStatus ( SDMAARM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | SDMA peripheral base address.       |
| <i>mask</i> | The stop status need to be cleared. |

### 25.7.18 static uint32\_t SDMA\_GetChannelPendStatus ( SDMAARM\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The pending status for all channels. Check the relevant bits for specific channel.

### 25.7.19 static void SDMA\_ClearChannelPendStatus ( SDMAARM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | SDMA peripheral base address.          |
| <i>mask</i> | The pending status need to be cleared. |

### 25.7.20 static uint32\_t SDMA\_GetErrorStatus ( SDMAARM\_Type \* *base* ) [inline], [static]

SDMA channel error flag is asserted while an incoming DMA request was detected and it triggers a channel that is already pending or being serviced. This probably means there is an overflow of data for that channel.

Parameters

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

Returns

The error status for all channels. Check the relevant bits for specific channel.

### 25.7.21 bool SDMA\_GetRequestSourceStatus ( SDMAARM\_Type \* *base*, uint32\_t *source* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | SDMA peripheral base address. |
| <i>source</i> | DMA request source number.    |

Returns

True means the request source is pending, otherwise not pending.

### 25.7.22 void SDMA\_CreateHandle ( sdma\_handle\_t \* *handle*, SDMAARM\_Type \* *base*, uint32\_t *channel*, sdma\_context\_data\_t \* *context* )

This function is called if using the transactional API for SDMA. This function initializes the internal state of the SDMA handle.

Parameters

|                |                                                                                                                                                                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | SDMA handle pointer. The SDMA handle stores callback function and parameters.                                                                                                                                                                                               |
| <i>base</i>    | SDMA peripheral base address.                                                                                                                                                                                                                                               |
| <i>channel</i> | SDMA channel number.                                                                                                                                                                                                                                                        |
| <i>context</i> | Context structure for the channel to download into SDMA. Users shall make sure the context located in a non-cacheable memory, or it will cause SDMA run fail. Users shall not touch the context contents, it only be filled by SDMA driver in SDMA_SubmitTransfer function. |

## Function Documentation

**25.7.23 void SDMA\_InstallIBDMemory ( *sdma\_handle\_t* \* *handle*,  
                          *sdma\_buffer\_descriptor\_t* \* *BDPool*, *uint32\_t* *BDCount* )**

This function is called after the SDMA\_CreateHandle to use multi-buffer feature.

Parameters

|                |                                                                          |
|----------------|--------------------------------------------------------------------------|
| <i>handle</i>  | SDMA handle pointer.                                                     |
| <i>BDPool</i>  | A memory pool to store BDs. It must be located in non-cacheable address. |
| <i>BDCount</i> | The number of BD slots.                                                  |

#### **25.7.24 void SDMA\_SetCallback ( *sdma\_handle\_t \* handle, sdma\_callback callback, void \* userData* )**

This callback is called in the SDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>handle</i>   | SDMA handle pointer.                   |
| <i>callback</i> | SDMA callback function pointer.        |
| <i>userData</i> | A parameter for the callback function. |

#### **25.7.25 void SDMA\_PreparesTransfer ( *sdma\_transfer\_config\_t \* config, uint32\_t srcAddr, uint32\_t destAddr, uint32\_t srcWidth, uint32\_t destWidth, uint32\_t bytesEachRequest, uint32\_t transferSize, uint32\_t eventSource, sdma\_peripheral\_t peripheral, sdma\_transfer\_type\_t type* )**

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

## Function Documentation

Parameters

|                         |                                                                                 |
|-------------------------|---------------------------------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type sdma_transfer_t.                       |
| <i>srcAddr</i>          | SDMA transfer source address.                                                   |
| <i>destAddr</i>         | SDMA transfer destination address.                                              |
| <i>srcWidth</i>         | SDMA transfer source address width(bytes).                                      |
| <i>destWidth</i>        | SDMA transfer destination address width(bytes).                                 |
| <i>bytesEachRequest</i> | SDMA transfer bytes per channel request.                                        |
| <i>transferSize</i>     | SDMA transfer bytes to be transferred.                                          |
| <i>eventSource</i>      | Event source number for the transfer, if use software trigger, just write 0.    |
| <i>peripheral</i>       | Peripheral type, used to decide if need to use some special scripts.            |
| <i>type</i>             | SDMA transfer type. Used to decide the correct SDMA script address in SDMA ROM. |

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.

### 25.7.26 void SDMA\_SubmitTransfer ( *sdma\_handle\_t \* handle*, *const sdma\_transfer\_config\_t \* config* )

This function submits the SDMA transfer request according to the transfer configuration structure.

Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>handle</i> | SDMA handle pointer.                              |
| <i>config</i> | Pointer to SDMA transfer configuration structure. |

### 25.7.27 void SDMA\_StartTransfer ( *sdma\_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> | SDMA handle pointer. |
|---------------|----------------------|

### 25.7.28 void SDMA\_StopTransfer ( *sdma\_handle\_t \* handle* )

This function disables the channel request to pause the transfer. Users can call [SDMA\\_StartTransfer\(\)](#) again to resume the transfer.

Parameters

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

### 25.7.29 void SDMA\_AbortTransfer ( *sdma\_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. |
|---------------|---------------------|

### 25.7.30 void SDMA\_HandleIRQ ( *sdma\_handle\_t \* handle* )

This function clears the interrupt flags and also handle the CCB for the channel.

Parameters

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

## Function Documentation

# Chapter 26

## SNVS\_HP: Secure Non-Volatile Storage

### 26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage (SNVS) module.

The SNVS module is designed to safely hold security-related data such as cryptographic key, time counter, monotonic counter, and general purpose security information. The SNVS includes a low power section, namely SNVS\_LP, that is battery-backed by the SVNS (or VBAT) power domain. This enables it to keep this data valid while continuing to increment the time counter when the power is lost 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. Specifically, 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 example, 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.

### 26.2 SNVS\_HP Driver Initialization and Configuration

The function [SNVS\\_HP\\_RTC\\_Init\(\)](#) initializes the SNVS with specified configurations. The function [SNVS\\_HP\\_RTC\\_GetDefaultConfig\(\)](#) gets the default configurations.

The function [SNVS\\_HP\\_RTC\\_Deinit\(\)](#) disables the SNVS RTC timer and disables the module clock.

### 26.3 Set & Get Datetime

The function [SNVS\\_HP\\_RTC\\_SetDatetime\(\)](#) sets the SNVS RTC date and time according to the given time structure described below.

```
typedef struct _snvs_hp_rtc_datetime
{
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
} snvs_hp_rtc_datetime_t;
```

The function [SNVS\\_HP\\_RTC\\_GetDatetime\(\)](#) reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

## Typical use case

### 26.4 Set & Get Alarm

The function [SNVS\\_HP\\_RTC\\_SetAlarm\(\)](#) sets the alarm time period in seconds. Users pass in the details in date & time format by using the datetime data structure.

The function [SNVS\\_HP\\_RTC\\_GetAlarm\(\)](#) reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

### 26.5 Start & Stop timer

The function [SNVS\\_HP\\_RTC\\_StartTimer\(\)](#) starts the SNVS\_HP RTC time counter.

The function [SNVS\\_HP\\_RTC\\_StopTimer\(\)](#) stops the SNVS\_HP RTC time counter.

### 26.6 Status

Provides functions to get and clear the SNVS\_HP status.

### 26.7 Interrupt

Provides functions to enable/disable SNVS\_HP interrupts and get current enabled interrupts.

### 26.8 Typical use case

#### 26.8.1 SNVS\_HP RTC example

Example to set the SNVS\_HP RTC current time and trigger an alarm.

```
int main(void)
{
    uint32_t sec;
    snvs_hp_rtc_datetime_t rtcDate;
    snvs_hp_rtc_config_t snvsRtcConfig;

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();
    /* Init SNVS */
    /*
     * snvsConfig->rtccalenable = false;
     * snvsConfig->rtccalvalue = 0U;
     * snvsConfig->srtccalenable = false;
     * snvsConfig->srtccalvalue = 0U;
     * snvsConfig->PIFreq = 0U;
     */
    SNVS_HP_RTC_GetDefaultConfig(&snvsRtcConfig);
    SNVS_HP_RTC_Init(SNVS, &snvsRtcConfig);

    PRINTF("SNVS HP example:\r\n");

    /* Set a start date time and start RT */
    rtcDate.year = 2014U;
    rtcDate.month = 12U;
    rtcDate.day = 25U;
    rtcDate.hour = 19U;
    rtcDate.minute = 0;
    rtcDate.second = 0;

    /* Set RTC time to default time and date and start the RTC */
```

```

SNVS_HP_RTC_SetDatetime(SNVS, &rtcDate);
SNVS_HP_RTC_StartTimer(SNVS);

/* Enable SNVS alarm interrupt */
SNVS_HP_RTC_EnableInterrupts(SNVS,
    kSNVS_RTC_AlarmInterruptEnable);

/* Enable at the NVIC */
EnableIRQ(EXAMPLE_SNVS IRQn);

PRINTF("Set up time to wake up an alarm.\r\n");
/* This loop will set the SNVS alarm */
while (1)
{
    busyWait = true;
    /* Get date time */
    SNVS_HP_RTC_GetDatetime(SNVS, &rtcDate);

    /* print default time */
    PRINTF("Current datetime: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n",
        rtcDate.year, rtcDate.month, rtcDate.day,
        rtcDate.hour, rtcDate.minute, rtcDate.second);

    /* Get alarm time from user */
    sec = 0;
    PRINTF("Please input the number of second to wait for alarm and press enter \r\n");
    PRINTF("The second must be positive value\r\n");

    while (sec < 1)
    {
        SCANF("%d", &sec);
    }

    SNVS_HP_RTC_GetDatetime(SNVS, &rtcDate);
    if ((rtcDate.second + sec) < 60)
    {
        rtcDate.second += sec;
    }
    else
    {
        rtcDate.minute += (rtcDate.second + sec) / 60U;
        rtcDate.second = (rtcDate.second + sec) % 60U;
    }

    SNVS_HP_RTC_SetAlarm(SNVS, &rtcDate);

    /* Get alarm time */
    SNVS_HP_RTC_GetAlarm(SNVS, &rtcDate);

    /* Print alarm time */
    PRINTF("Alarm will occur at: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n",
        rtcDate.year, rtcDate.month, rtcDate.day,
        rtcDate.hour, rtcDate.minute, rtcDate.second);

    /* Wait until alarm occurs */
    while (busyWait)
    {

    }

    PRINTF("\r\n Alarm occurs !!!! ");
}
}

```

## Data Structures

- struct `snvs_hp_rtc_datetime_t`

## Typical use case

Structure is used to hold the date and time. [More...](#)

- struct `snvs_hp_rtc_config_t`  
SNVS config structure. [More...](#)

## Enumerations

- enum `snvs_hp_interrupt_enable_t` {  
`kSNVS_RTC_PeriodicInterruptEnable` = 1U,  
`kSNVS_RTC_AlarmInterruptEnable` = 2U }  
*List of SNVS interrupts.*
- enum `snvs_hp_status_flags_t` {  
`kSNVS_RTC_PeriodicInterruptFlag` = 1U,  
`kSNVS_RTC_AlarmInterruptFlag` = 2U }  
*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`  
*Range from 1 to 31 (depending on month).*
- `uint8_t snvs_hp_rtc_datetime_t::hour`  
*Range from 0 to 23.*
- `uint8_t snvs_hp_rtc_datetime_t::minute`  
*Range from 0 to 59.*
- `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.*

## Interrupt Interface

- void [SNVS\\_HP\\_RTC\\_EnableInterrupts](#) (SNVS\_Type \*base, uint32\_t mask)
 

*Enables the selected SNVS interrupts.*
- 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.*
- 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.*

## 26.9 Data Structure Documentation

### 26.9.1 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.*

## Function Documentation

- `uint8_t minute`  
*Range from 0 to 59.*
- `uint8_t second`  
*Range from 0 to 59.*

### 26.9.2 `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.*

## 26.10 Enumeration Type Documentation

### 26.10.1 `enum snvs_hp_interrupt_enable_t`

Enumerator

`kSNVS_RTC_PeriodicInterruptEnable` RTC periodic interrupt.  
`kSNVS_RTC_AlarmInterruptEnable` RTC time alarm.

### 26.10.2 `enum snvs_hp_status_flags_t`

Enumerator

`kSNVS_RTC_PeriodicInterruptFlag` RTC periodic interrupt flag.  
`kSNVS_RTC_AlarmInterruptFlag` RTC time alarm flag.

## 26.11 Function Documentation

### 26.11.1 `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.

## Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | SNVS peripheral base address                        |
| <i>config</i> | Pointer to the user's SNVS configuration structure. |

**26.11.2 void SNVS\_HP\_RTC\_Deinit ( SNVS\_Type \* *base* )**

## Parameters

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

**26.11.3 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. |
|---------------|-----------------------------------------------------|

**26.11.4 status\_t SNVS\_HP\_RTC\_SetDatetime ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_datetime\_t \* *datetime* )**

## Parameters

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

## Function Documentation

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <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

### 26.11.5 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. |

### 26.11.6 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

### 26.11.7 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. |

#### 26.11.8 void SNVS\_HP\_RTC\_EnableInterrupts ( **SNVS\_Type** \* *base*, **uint32\_t** *mask* )

Parameters

|             |                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                           |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration ::snvs_interrupt_enable_t |

#### 26.11.9 void SNVS\_HP\_RTC\_DisableInterrupts ( **SNVS\_Type** \* *base*, **uint32\_t** *mask* )

Parameters

|             |                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                           |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration ::snvs_interrupt_enable_t |

#### 26.11.10 **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

#### 26.11.11 **uint32\_t** SNVS\_HP\_RTC\_GetStatusFlags ( **SNVS\_Type** \* *base* )

## Function Documentation

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

### 26.11.12 void SNVS\_HP\_RTC\_ClearStatusFlags ( SNVS\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                        |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration ::snvs_status_flags_t |

### 26.11.13 static void SNVS\_HP\_RTC\_StartTimer ( SNVS\_Type \* *base* ) [inline], [static]

Parameters

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

### 26.11.14 static void SNVS\_HP\_RTC\_StopTimer ( SNVS\_Type \* *base* ) [inline], [static]

Parameters

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

## 26.12 Variable Documentation

26.12.1 `uint16_t snvs_hp_rtc_datetime_t::year`

26.12.2 `uint8_t snvs_hp_rtc_datetime_t::month`

26.12.3 `uint8_t snvs_hp_rtc_datetime_t::day`

26.12.4 `uint8_t snvs_hp_rtc_datetime_t::hour`

26.12.5 `uint8_t snvs_hp_rtc_datetime_t::minute`

26.12.6 `uint8_t snvs_hp_rtc_datetime_t::second`

## Variable Documentation

# Chapter 27

## SRC: System Reset Controller Driver

### 27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Reset Controller (SRC) module.

The System Reset Controller (SRC) controls the reset and boot operation of the SoC. It is responsible for the generation of all reset signals and boot decoding. The reset controller determines the source and the type of reset, such as POR, WARM, COLD, and performs the necessary reset qualification and stretching sequences. Based on the type of reset, the reset logic generates the reset sequence for the entire IC.

### Enumerations

- enum `_src_reset_status_flags` {  
    `kSRC_WarmBootIndicationFlag` = `SRC_SRCSR_WBI_MASK`,  
    `kSRC_TemperatureSensorResetFlag` = `SRC_SRCSR_TS_R_RESET_B_MASK`,  
    `kSRC_Wdog3ResetFlag` = `SRC_SRCSR_WDOG3_RST_B_MASK`,  
    `kSRC_JTAGSoftwareResetFlag` = `SRC_SRCSR_SJC_MASK`,  
    `kSRC_JTAGGeneratedResetFlag` = `SRC_SRCSR_JTAG_MASK`,  
    `kSRC_WatchdogResetFlag` = `SRC_SRCSR_WDOG_MASK`,  
    `kSRC_IppUserResetFlag` = `SRC_SRCSR_IPP_USER_RESET_B_MASK`,  
    `kSRC_CsuResetFlag` = `SRC_SRCSR_CSU_RESET_B_MASK`,  
    `kSRC_IppResetPinFlag` = `SRC_SRCSR_IPP_RESET_B_MASK` }  
        *SRC reset status flags.*
- enum `_src_status_flags` { `kSRC_Core0WdogResetReqFlag` }  
        *SRC interrupt status flag.*
- enum `src_mix_reset_stretch_cycles_t` {  
    `kSRC_MixResetStretchCycleAlt0` = `0U`,  
    `kSRC_MixResetStretchCycleAlt1` = `1U`,  
    `kSRC_MixResetStretchCycleAlt2` = `2U`,  
    `kSRC_MixResetStretchCycleAlt3` = `3U` }  
        *Selection of SoC mix power reset stretch.*
- enum `src_wdog3_reset_option_t` {  
    `kSRC_Wdog3ResetOptionAlt0` = `0U`,  
    `kSRC_Wdog3ResetOptionAlt1` = `1U` }  
        *Selection of WDOG3 reset option.*
- 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.*

## Overview

## Functions

- static void [SRC\\_EnableWDOG3Reset](#) (SRC\_Type \*base, bool enable)  
*Enable the WDOG3 reset.*
- static void [SRC\\_SetMixResetStretchCycles](#) (SRC\_Type \*base, [src\\_mix\\_reset\\_stretch\\_cycles\\_t](#) option)  
*Set the mix power up reset stretch mix reset width.*
- static void [SRC\\_EnableCoreDebugResetAfterPowerGate](#) (SRC\_Type \*base, bool enable)  
*Debug reset would be asserted after power gating event.*
- static void [SRC\\_SetWdog3ResetOption](#) (SRC\_Type \*base, [src\\_wdog3\\_reset\\_option\\_t](#) option)  
*Set the Wdog3\_RST\_B option.*
- static void [SRC\\_DoSoftwareResetARMCoreDebug](#) (SRC\_Type \*base)  
*Software reset for debug of arm platform only.*
- static bool [SRC\\_GetSoftwareResetARMCoreDebugDone](#) (SRC\_Type \*base)  
*Check if the software reset for debug of arm platform only is done.*
- 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\\_AssertEIMReset](#) (SRC\_Type \*base, bool enable)  
*Assert the EIM reset.*
- static void [SRC\\_EnableWDOGReset](#) (SRC\_Type \*base, bool enable)  
*Enable the WDOG Reset in SRC.*
- static void [SRC\\_SetWarmResetBypassCount](#) (SRC\_Type \*base, [src\\_warm\\_reset\\_bypass\\_count\\_t](#) option)  
*Set the delay count of waiting MMDC's acknowledge.*
- static void [SRC\\_EnableWarmReset](#) (SRC\_Type \*base, bool enable)  
*Enable the WARM reset.*
- static uint32\_t [SRC\\_GetStatusFlags](#) (SRC\_Type \*base)  
*Get interrupt status flags.*
- static uint32\_t [SRC\\_GetBootModeWord1](#) (SRC\_Type \*base)  
*Get the boot mode register 1 value.*
- static uint32\_t [SRC\\_GetBootModeWord2](#) (SRC\_Type \*base)  
*Get the boot mode register 2 value.*
- static void [SRC\\_SetWarmBootIndication](#) (SRC\_Type \*base, bool enable)  
*Set the warm boot indication flag.*
- static uint32\_t [SRC\\_GetResetStatusFlags](#) (SRC\_Type \*base)  
*Get the status flags of SRC.*
- void [SRC\\_ClearResetStatusFlags](#) (SRC\_Type \*base, uint32\_t flags)  
*Clear the status flags of SRC.*
- static void [SRC\\_SetGeneralPurposeRegister](#) (SRC\_Type \*base, uint32\_t index, uint32\_t value)  
*Set value to general purpose registers.*
- static uint32\_t [SRC\\_GetGeneralPurposeRegister](#) (SRC\_Type \*base, uint32\_t index)  
*Get the value from general purpose registers.*

## Driver version

- #define [FSL\\_SRC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*SRC driver version 2.0.0.*

## 27.2 Macro Definition Documentation

### 27.2.1 #define FSL\_SRC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 27.3 Enumeration Type Documentation

### 27.3.1 enum \_src\_reset\_status\_flags

Enumerator

***kSRC\_WarmBootIndicationFlag*** WARM boot indication shows that WARM boot was initiated by software.

***kSRC\_TemperatureSensorResetFlag*** Indicates whether the reset was the result of software reset from on-chip Temperature Sensor. Temperature Sensor Interrupt 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\_IppResetPinFlag*** Indicates whether reset was the result of ipp\_reset\_b pin (Power-up sequence).

### 27.3.2 enum \_src\_status\_flags

Enumerator

***kSRC\_Core0WdogResetReqFlag*** WDOG reset request from core0. Read-only status bit.

### 27.3.3 enum src\_mix\_reset\_stretch\_cycles\_t

This type defines the SoC mix (Audio, ENET, uSDHC, EIM, QSPI, OCRAM, MMDC, etc) power up reset stretch mix reset width with the optional count of cycles

Enumerator

***kSRC\_MixResetStretchCycleAlt0*** mix reset width is 1 x 88 ipg\_cycle cycles.

***kSRC\_MixResetStretchCycleAlt1*** mix reset width is 2 x 88 ipg\_cycle cycles.

## Function Documentation

*kSRC\_MixResetStretchCycleAlt2* mix reset width is 3 x 88 ipg\_cycle cycles.

*kSRC\_MixResetStretchCycleAlt3* mix reset width is 4 x 88 ipg\_cycle cycles.

### 27.3.4 enum src\_wdog3\_reset\_option\_t

Enumerator

*kSRC\_Wdog3ResetOptionAlt0* Wdog3\_rst\_b asserts M4 reset (default).

*kSRC\_Wdog3ResetOptionAlt1* Wdog3\_rst\_b asserts global reset.

### 27.3.5 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.

## 27.4 Function Documentation

### 27.4.1 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.     |

### 27.4.2 static void SRC\_SetMixResetStretchCycles ( *SRC\_Type* \* *base*, *src\_mix\_reset\_stretch\_cycles\_t option* ) [inline], [static]

Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | SRC peripheral base address.                                            |
| <i>option</i> | Setting option, see to <a href="#">src_mix_reset_stretch_cycles_t</a> . |

#### 27.4.3 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.     |

#### 27.4.4 static void SRC\_SetWdog3ResetOption ( SRC\_Type \* *base*, src\_wdog3\_reset\_option\_t *option* ) [inline], [static]

Parameters

|               |                                                                   |
|---------------|-------------------------------------------------------------------|
| <i>base</i>   | SRC peripheral base address.                                      |
| <i>option</i> | Setting option, see to <a href="#">src_wdog3_reset_option_t</a> . |

#### 27.4.5 static void SRC\_DoSoftwareResetARMCoreDebug ( SRC\_Type \* *base* ) [inline], [static]

Parameters

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

#### 27.4.6 static bool SRC\_GetSoftwareResetARMCoreDebugDone ( SRC\_Type \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

**27.4.7 static void SRC\_DoSoftwareResetARMCore0 ( SRC\_Type \* *base* )  
[inline], [static]**

Parameters

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

**27.4.8 static bool SRC\_GetSoftwareResetARMCore0Done ( SRC\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

If the reset is done.

**27.4.9 static void SRC\_AssertEIMReset ( SRC\_Type \* *base*, bool *enable* )  
[inline], [static]**

EIM reset is needed in order to reconfigure the EIM chip select. The software reset bit must de-asserted since this is not self-refresh.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SRC peripheral base address. |
| <i>enable</i> | Make the assertion or not.   |

**27.4.10 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.

## Function Documentation

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SRC peripheral base address. |
| <i>enable</i> | Enable the reset or not.     |

### 27.4.11 static void SRC\_SetWarmResetBypassCount ( SRC\_Type \* *base*, src\_warm\_reset\_bypass\_count\_t *option* ) [inline], [static]

This function would define the 32KHz clock cycles to count before bypassing the MMDC acknowledge for WARM reset. If the MMDC acknowledge is not asserted before this counter is elapsed, a COLD reset will be initiated.

Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | SRC peripheral base address.                                                       |
| <i>option</i> | The option of setting mode, see to <a href="#">src_warm_reset_bypass_count_t</a> . |

### 27.4.12 static void SRC\_EnableWarmReset ( SRC\_Type \* *base*, bool *enable* ) [inline], [static]

Only when the WARM reset is enabled, the WARM reset requests would be served by WARM reset. Otherwise, all the WARM reset sources would generate COLD reset.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | SRC peripheral base address.  |
| <i>enable</i> | Enable the WARM reset or not. |

### 27.4.13 static uint32\_t SRC\_GetStatusFlags ( SRC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Mask value of status flags. See to [\\$\\_src\\_status\\_flags](#).

**27.4.14 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.

## Function Documentation

Parameters

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

Returns

status of BOOT\_CFGx pins of the chip.

### 27.4.15 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.

### 27.4.16 static void SRC\_SetWarmBootIndication( SRC\_Type \* *base*, bool *enable* ) [inline], [static]

WARM boot indication shows that WARM boot was initiated by software. This indicates to the software that it saved the needed information in the memory before initiating the WARM reset. In this case, software will set this bit to '1', before initiating the WARM reset. The warm\_boot bit should be used as indication only after a warm\_reset sequence. Software should clear this bit after warm\_reset to indicate that the next warm\_reset is not performed with warm\_boot.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SRC peripheral base address. |
| <i>enable</i> | Assert the flag or not.      |

### 27.4.17 static uint32\_t SRC\_GetResetStatusFlags( SRC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Mask value of status flags, see to [\\_src\\_reset\\_status\\_flags](#).

#### 27.4.18 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> . |

#### 27.4.19 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 arbitrary 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.                                           |

#### 27.4.20 static uint32\_t SRC\_GetGeneralPurposeRegister ( **SRC\_Type** \* *base*, **uint32\_t** *index* ) [inline], [static]

Parameters

## Function Documentation

|              |                                                                            |
|--------------|----------------------------------------------------------------------------|
| <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 28

## TSC: Touch Screen Controller Driver

### 28.1 Overview

The MCUXpresso SDK provides Peripheral driver for the Touch Screen Controller(TSC) module of MCUXpresso SDK devices.

### 28.2 Typical use case

#### 28.2.1 4-wire Polling Configuration

```
// ...

tsc_config_t k_tscConfig;

BOARD_InitHardware();

PRINTF("TSC fourWireModePolling Example Start!\r\n");

ADC_Configuration();

TSC_GetDefaultConfig(&k_tscConfig);
k_tscConfig.enableAutoMeasure = true;
TSC_Init(DEMO_TSC_BASE, &k_tscConfig);
TSC_EnableInterruptSignals(DEMO_TSC_BASE, kTSC_ValidSignalEnable);
PRINTF("Please touch screen.\r\n");

while (1)
{
    TSC_SoftwareReset(DEMO_TSC_BASE);
    TSC_StartSenseDetection(DEMO_TSC_BASE);
    while ((TSC_GetStatusFlags(DEMO_TSC_BASE) & kTSC_StateMachineFlag) != kTSC_IdleState)
    {
    }
    if ((TSC_GetInterruptStatusFlags(DEMO_TSC_BASE) & kTSC_ValidSignalFlag) == kTSC_ValidSignalFlag)
    {
        TSC_ClearInterruptStatusFlags(DEMO_TSC_BASE, kTSC_ValidSignalFlag);
        PRINTF("x = %d, y = %d\r\n", TSC_GetMeasureValue(TSC,
            kTSC_XCoordinateValueSelection),
            TSC_GetMeasureValue(TSC, kTSC_YCoordinateValueSelection));
    }
}

// ...

void ADC_Configuration(void)
{
    adc_5hc_config_t k_adcConfig;
    adc_5hc_channel_config_t k_adcChannelConfig;

    ADC_5HC_GetDefaultConfig(&k_adcConfig);
    ADC_5HC_Init(DEMO_ADC_BASE, &k_adcConfig);
    ADC_5HC_EnableHardwareTrigger(DEMO_ADC_BASE, true);

    k_adcChannelConfig.channelNumber = 1U;
    k_adcChannelConfig.enableInterruptOnConversionCompleted = false;
```

## Typical use case

```
ADC_5HC_SetChannelConfig(DEMO_ADC_BASE, 3U, &k_adcChannelConfig);
k_adcChannelConfig.channelNumber = 3U;
ADC_5HC_SetChannelConfig(DEMO_ADC_BASE, 1U, &k_adcChannelConfig);

if (kStatus_Success == ADC_5HC_DoAutoCalibration(DEMO_ADC_BASE))
{
    PRINTF("ADC_5HC_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC_5HC_DoAutoCalibration() Failed.\r\n");
}
}
```

### 28.2.2 4-wire Interrupt Configuration

```
// ...

tsc_config_t k_tscConfig;

BOARD_InitHardware();

PRINTF("TSC fourWireModeInterrupt Example Start!\r\n");

ADC_Configuration();
GIC_Configuration();

TSC_GetDefaultConfig(&k_tscConfig);
TSC_Init(DEMO_TSC_BASE, &k_tscConfig);
TSC_EnableInterruptSignals(DEMO_TSC_BASE,
                           kTSC_ValidSignalEnable |
                           kTSC_MeasureSignalEnable | kTSC_DetectSignalEnable);
TSC_EnableInterrupts(DEMO_TSC_BASE, kTSC_MeasureInterruptEnable |
                     kTSC_DetectInterruptEnable);
PRINTF("Please touch screen.\r\n");

while (1)
{
    TSC_SoftwareReset(DEMO_TSC_BASE);
    TSC_StartSenseDetection(DEMO_TSC_BASE);
    while ((TSC_GetStatusFlags(DEMO_TSC_BASE) & kTSC_StateMachineFlag) != kTSC_IdleState)
    {
    }
    if (g_tscTouch)
    {
        g_tscTouch = false;
        PRINTF("x = %d, y = %d\r\n", TSC_GetMeasureValue(DEMO_TSC_BASE,
   kTSC_XCoordinateValueSelection),
               TSC_GetMeasureValue(DEMO_TSC_BASE, kTSC_YCoordinateValueSelection));
    }
}

// ...

void ADC_Configuration(void)
{
    adc_5hc_config_t k_adcConfig;
    adc_5hc_channel_config_t k_adcChannelConfig;

    ADC_5HC_GetDefaultConfig(&k_adcConfig);
    ADC_5HC_Init(DEMO_ADC_BASE, &k_adcConfig);
    ADC_5HC_EnableHardwareTrigger(DEMO_ADC_BASE, true);
```

```

k_adcChannelConfig.channelNumber = 1U; /* Channel1 is ynlr port. */
k_adcChannelConfig.enableInterruptOnConversionCompleted = false;
ADC_5HC_SetChannelConfig(DEMO_ADC_BASE, 3U, &k_adcChannelConfig);
k_adcChannelConfig.channelNumber = 3U; /* Channel3 is xnur port. */
ADC_5HC_SetChannelConfig(DEMO_ADC_BASE, 1U, &k_adcChannelConfig);

if (kStatus_Success == ADC_5HC_DoAutoCalibration(DEMO_ADC_BASE))
{
    PRINTF("ADC_5HC_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC_5HC_DoAutoCalibration() Failed.\r\n");
}

void GIC_Configuration(void)
{
    GIC_EnableIRQ(TSC_IRQn);
}

void EXAMPLE_TSC_IRQHandler(void)
{
    if ((TSC_GetInterruptStatusFlags(DEMO_TSC_BASE) & kTSC_DetectSignalFlag) == kTSC_DetectSignalFlag)
    {
        TSC_ClearInterruptStatusFlags(DEMO_TSC_BASE, kTSC_DetectSignalFlag);
        TSC_StartMeasure(DEMO_TSC_BASE);
    }
    else
    {
        if ((TSC_GetInterruptStatusFlags(DEMO_TSC_BASE) & kTSC_ValidSignalFlag) == kTSC_ValidSignalFlag)
        {
            TSC_ClearInterruptStatusFlags(DEMO_TSC_BASE, kTSC_ValidSignalFlag);
            g_tscTouch = true;
        }
        TSC_ClearInterruptStatusFlags(DEMO_TSC_BASE, kTSC_MeasureSignalFlag);
    }
}
}

```

## 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_coordinate_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_ADCConversionValueFlag`}  
        *@ 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.*

## 28.3 Data Structure Documentation

### 28.3.1 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.*

## 28.4 Macro Definition Documentation

### 28.4.1 `#define FSL_TSC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

Version 2.0.0.

## 28.5 Enumeration Type Documentation

### 28.5.1 `enum tsc_detection_mode_t`

Enumerator

`kTSC_Detection4WireMode` 4-Wire Detection Mode.

`kTSC_Detection5WireMode` 5-Wire Detection Mode.

### 28.5.2 `enum tsc_corrdinate_value_selection_t`

Enumerator

`kTSC_XCoordinateValueSelection` X coordinate value is selected.

`kTSC_YCoordinateValueSelection` Y coordinate value is selected.

### 28.5.3 `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.

#### 28.5.4 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.

#### 28.5.5 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.

#### 28.5.6 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.

#### 28.5.7 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.

### 28.5.8 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.

### 28.5.9 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.

### 28.5.10 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.

### 28.5.11 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.

### 28.5.12 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.

## 28.6 Function Documentation

### 28.6.1 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. |

### 28.6.2 void TSC\_Deinit ( TSC\_Type \* *base* )

## Variable Documentation

Parameters

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

### 28.7 Variable Documentation

#### 28.7.1 bool tsc\_config\_t::enableAutoMeasure

It indicates after detect touch, whether automatic start measurement

#### 28.7.2 uint32\_t tsc\_config\_t::measureDelayTime

If measure delay time is too short, maybe it would have an undesired effect on measure value.

#### 28.7.3 uint32\_t tsc\_config\_t::prechargeTime

It is a preparation for detection stage. Pre-charge time must be 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).

#### 28.7.4 tsc\_detection\_mode\_t tsc\_config\_t::detectionMode

See "tsc\_detection\_mode\_t".

# Chapter 29

## USDHC: ultra Secured Digital Host Controller Driver

### 29.1 Overview

The MCUXpresso SDK provides a peripheral driver for the ultra Secured Digital Host Controller (USDHC) module of MCUXpresso SDK/i.MX devices.

### 29.2 Typical use case

#### 29.2.1 USDHC Operation

```
/* Initializes the USDHC. */
usdhcConfig->dataTimeout = 0xFU;
usdhcConfig->endianMode = kUSDHC_EndianModeLittle;
usdhcConfig->readWatermarkLevel = 0x80U;
usdhcConfig->writeWatermarkLevel = 0x80U;
usdhcConfig->readBurstLen = 16U;
usdhcConfig->writeBurstLen = 16U;
USDHC_Init(BOARD_USDHC_BASEADDR, usdhcConfig);

/* Fills state in the card driver. */
card->usdhcBase = BOARD_USDHC_BASEADDR;
card->usdhcSourceClock = CLOCK_GetFreq(BOARD_USDHC_CLKSRC);
card->usdhcTransfer = usdhc_transfer_function;

/* Initializes the card. */
if (SD_Init(card))
{
    PRINTF("\r\nSD card init failed.\r\n");
}

PRINTF("\r\nRead/Write/Erase the card continuously until it encounters error.....\r\n");
while (true)
{
    if (kStatus_Success != SD_WriteBlocks(card, g_dataWrite, DATA_BLOCK_START, DATA_BLOCK_COUNT))
    {
        PRINTF("Write multiple data blocks failed.\r\n");
    }
    if (kStatus_Success != SD_ReadBlocks(card, g_dataRead, DATA_BLOCK_START, DATA_BLOCK_COUNT))
    {
        PRINTF("Read multiple data blocks failed.\r\n");
    }

    if (kStatus_Success != SD_EraseBlocks(card, DATA_BLOCK_START, DATA_BLOCK_COUNT))
    {
        PRINTF("Erase multiple data blocks failed.\r\n");
    }
}

SD_Deinit(card);
```

### Data Structures

- struct [usdhc\\_adma2\\_descriptor\\_t](#)  
*Defines the ADMA2 descriptor structure. [More...](#)*

## Typical use case

- 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.*
- #define `USDHC_ADMA1_LENGTH_ALIGN` (4096U)  
*The alignment size for LENGTH field in ADMA1's descriptor.*
- #define `USDHC_ADMA2_ADDRESS_ALIGN` (4U)  
*The alignment size for ADDRESS field in ADMA2's descriptor.*
- #define `USDHC_ADMA2_LENGTH_ALIGN` (4U)  
*The alignment size for LENGTH filed in ADMA2's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT` (12U)  
*The bit shift for ADDRESS filed in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK` (0xFFFFFU)  
*The bit mask for ADDRESS field in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT` (12U)  
*The bit shift for LENGTH filed in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK` (0xFFFFU)  
*The mask for LENGTH field in ADMA1's descriptor.*
- #define `USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (`USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK - 3U`)  
*The maximum value of LENGTH filed in ADMA1's descriptor.*
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT` (16U)  
*The bit shift for LENGTH field in ADMA2's descriptor.*
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK` (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_DMADataAddrNotAlign = MAKE_STATUS(kStatusGroup_USDHC, 4U),`  
`kStatus_USDHC_ReTuningRequest = MAKE_STATUS(kStatusGroup_USDHC, 5U),`  
`kStatus_USDHC_TuningError = MAKE_STATUS(kStatusGroup_USDHC, 6U) }`  
*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.*

## Typical use case

- 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 {  
    kUSDHC\_CommandInhibitFlag = USDHC\_PRES\_STATE\_CIHB\_MASK,  
    kUSDHC\_DataInhibitFlag = USDHC\_PRES\_STATE\_CDIHB\_MASK,  
    kUSDHC\_DataLineActiveFlag = USDHC\_PRES\_STATE\_DLAI\_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\_DLSL\_SHIFT),  
    kUSDHC\_Data1LineLevelFlag = (1U << (USDHC\_PRES\_STATE\_DLSL\_SHIFT + 1U)),  
    kUSDHC\_Data2LineLevelFlag = (1U << (USDHC\_PRES\_STATE\_DLSL\_SHIFT + 2U)),  
    kUSDHC\_Data3LineLevelFlag = (1U << (USDHC\_PRES\_STATE\_DLSL\_SHIFT + 3U)),  
    kUSDHC\_Data4LineLevelFlag = (1U << (USDHC\_PRES\_STATE\_DLSL\_SHIFT + 4U)),  
    kUSDHC\_Data5LineLevelFlag = (1U << (USDHC\_PRES\_STATE\_DLSL\_SHIFT + 5U)),  
    kUSDHC\_Data6LineLevelFlag = (1U << (USDHC\_PRES\_STATE\_DLSL\_SHIFT + 6U)),  
    kUSDHC\_Data7LineLevelFlag = (1U << (USDHC\_PRES\_STATE\_DLSL\_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_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_TuningErrorFlag = USDHC_INT_STATUS_TNE_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_AUTO_CMD12_ERR_STATUS_EXECUTE_TUNING_M-
ASK,
kUSDHC_TuningSampleClockSel }
    standard tuning flag
• enum _usdhc_adma_error_status_flag {
    kUSDHC_AdmaLenghMismatchFlag = USDHC_ERR_STATUS_ADMALME_MASK,
    kUSDHC_AdmaDescriptorErrorFlag = USDHC_ERR_STATUS_ADMADCE_MASK }
    ADMA error status flag mask.
• enum usdhc_adma_error_state_t {
    kUSDHC_AdmaErrorStateStopDma = 0x00U,
    kUSDHC_AdmaErrorStateFetchDescriptor = 0x01U,
    kUSDHC_AdmaErrorStateChangeAddress = 0x02U,
    kUSDHC_AdmaErrorStateTransferData = 0x03U }
    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_FEVTCCEBE_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_FEVTDDEBE_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 }
```

*The command type.*

- enum `usdhc_card_response_type_t` {
 

```
kCARD_ResponseNone = 0U,
kCARD_ResponseR1 = 1U,
kCARD_ResponseR1b = 2U,
kCARD_ResponseR2 = 3U,
kCARD_ResponseR3 = 4U,
kCARD_ResponseR4 = 5U,
kCARD_ResponseR5 = 6U,
kCARD_ResponseR5b = 7U,
kCARD_ResponseR6 = 8U,
kCARD_ResponseR7 = 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_burst_len_t`{  
    kUSDHC\_EnBurstLenForINCR = 0x01U,  
    kUSDHC\_EnBurstLenForINCR416 = 0x02U,  
    kUSDHC\_EnBurstLenForINCR416WRAP = 0x04U }

*dma transfer burst len config.*

## Driver version

- #define `FSL_USDHCI_DRIVER_VERSION` (MAKE\_VERSION(2U, 1U, 1U))  
*Driver version 2.1.1.*

## Initialization and deinitialization

- void `USDHCI_Init` (USDHCI\_Type \*base, const `usdhc_config_t` \*config)  
*USDHCI module initialization function.*
- void `USDHCI_Deinit` (USDHCI\_Type \*base)  
*Deinitializes the USDHCI.*
- bool `USDHCI_Reset` (USDHCI\_Type \*base, uint32\_t mask, uint32\_t timeout)  
*Resets the USDHCI.*

## DMA Control

- status\_t `USDHCI_SetAdmaTableConfig` (USDHCI\_Type \*base, `usdhc_adma_config_t` \*dmaConfig, `usdhc_data_t` \*dataConfig, uint32\_t flags)  
*Sets the ADMA descriptor table configuration.*

## 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)  
*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 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.*

## Typical use case

- 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.*
- void **USDHC\_SetMmcBootConfig** (USDHC\_Type \*base, const usdhc\_boot\_config\_t \*config)  
*Configures the MMC boot feature.*
- 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,*
- static void **USDHC\_EnableDDRMode** (USDHC\_Type \*base, bool enable, uint32\_t nibblePos)  
*the enable/disable DDR mode*
- void **USDHC\_EnableAutoTuningForCmdAndData** (USDHC\_Type \*base)  
*the enable/disable HS400 mode*
- 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, HS200 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*

## Transactional

- 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.*

## 29.3 Data Structure Documentation

### 29.3.1 struct usdhc\_adma2\_descriptor\_t

#### Data Fields

- `uint32_t attribute`  
*The control and status field.*
- `const uint32_t * address`  
*The address field.*

### 29.3.2 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)*

### 29.3.3 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`

## Data Structure Documentation

- bool `enableBootAck`  
*Stop at block gap value of automatic mode.*
- bool `enableBoot`  
*Enable or disable boot ACK.*
- bool `enableAutoStopAtBlockGap`  
*Enable or disable fast boot.*
- bool `enableAutoStopAtBlockGap`  
*Enable or disable auto stop at block gap function in boot period.*

### 29.3.3.0.0.63 Field Documentation

#### 29.3.3.0.0.63.1 `uint32_t usdhc_boot_config_t::ackTimeoutCount`

The available range is 0 ~ 15.

#### 29.3.3.0.0.63.2 `usdhc_boot_mode_t usdhc_boot_config_t::bootMode`

#### 29.3.3.0.0.63.3 `uint32_t usdhc_boot_config_t::blockCount`

Available range is 0 ~ 65535.

## 29.3.4 `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.*

### 29.3.4.0.0.64 Field Documentation

#### 29.3.4.0.0.64.1 `uint8_t usdhc_config_t::readWatermarkLevel`

Available range is 1 ~ 128.

#### 29.3.4.0.0.64.2 `uint8_t usdhc_config_t::writeWatermarkLevel`

Available range is 1 ~ 128.

### 29.3.5 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**  
*Enable to ignore error event to read/write all the data.*
- bool **executeTuning**  
*execute tuning flag*
- 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.*

### 29.3.6 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.*

## Data Structure Documentation

### 29.3.7 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.*

### 29.3.8 struct usdhc\_transfer\_t

#### Data Fields

- usdhc\_data\_t \* data  
*Data to transfer.*
- usdhc\_command\_t \* command  
*Command to send.*

### 29.3.9 struct usdhc\_transfer\_callback\_t

#### Data Fields

- void(\* CardInserted )(void)  
*Card inserted occurs when DAT3/CD pin is for card detect.*
- void(\* CardRemoved )(void)  
*Card removed occurs.*
- void(\* SdioInterrupt )(void)  
*SDIO card interrupt occurs.*
- void(\* SdioBlockGap )(void)  
*SDIO card stopped at block gap occurs.*
- void(\* TransferComplete )(USDHC\_Type \*base, usdhc\_handle\_t \*handle, status\_t status, void \*userData)  
*Transfer complete callback.*
- void(\* ReTuning )(void)  
*handle the re-tuning*

### 29.3.10 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.*
- `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.*

### 29.3.11 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.*

## 29.4 Macro Definition Documentation

### 29.4.1 #define FSL\_USDHC\_DRIVER\_VERSION (MAKE\_VERSION(2U, 1U, 1U))

## Enumeration Type Documentation

### 29.5 Typedef Documentation

29.5.1 `typedef uint32_t usdhc_adma1_descriptor_t`

29.5.2 `typedef status_t(* usdhc_transfer_function_t)(USDHC_Type *base, usdhc_transfer_t *content)`

### 29.6 Enumeration Type Documentation

29.6.1 `enum _usdhc_status`

Enumerator

*kStatus\_USDHC\_BusyTransferring* Transfer is on-going.

*kStatus\_USDHC\_PrepAdmaDescriptorFailed* Set DMA descriptor failed.

*kStatus\_USDHC\_SendCommandFailed* Send command failed.

*kStatus\_USDHC\_TransferDataFailed* Transfer data failed.

*kStatus\_USDHC\_DMADataAddrNotAlign* data address not align

*kStatus\_USDHC\_ReTuningRequest* re-tuning request

*kStatus\_USDHC\_TuningError* tuning error

29.6.2 `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

29.6.3 `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.

#### 29.6.4 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.

#### 29.6.5 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.

#### 29.6.6 enum \_usdhc\_present\_status\_flag

Enumerator

*kUSDHC\_CommandInhibitFlag* Command inhibit.  
*kUSDHC\_DataInhibitFlag* Data inhibit.  
*kUSDHC\_DataLineActiveFlag* Data line active.

## Enumeration Type Documentation

*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.

### 29.6.7 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\_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\_TuningErrorFlag* SDR104 tuning error flag.  
*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.

### 29.6.8 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.

### 29.6.9 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 sampleing clock

### 29.6.10 enum \_usdhc\_adma\_error\_status\_flag

Enumerator

*kUSDHC\_AdmaLengthMismatchFlag* Length mismatch error.  
*kUSDHC\_AdmaDescriptorErrorFlag* Descriptor error.

### 29.6.11 enum usdhc\_adma\_error\_state\_t

This state is the detail state when ADMA error has occurred.

Enumerator

*kUSDHC\_AdmaErrorStateStopDma* Stop DMA.

## Enumeration Type Documentation

*kUSDHC\_AdmaErrorStateFetchDescriptor* Fetch descriptor.  
*kUSDHC\_AdmaErrorStateChangeAddress* Change address.  
*kUSDHC\_AdmaErrorStateTransferData* Transfer data.

### 29.6.12 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.

### 29.6.13 enum usdhc\_data\_bus\_width\_t

Enumerator

*kUSDHC\_DataBusWidth1Bit* 1-bit mode  
*kUSDHC\_DataBusWidth4Bit* 4-bit mode  
*kUSDHC\_DataBusWidth8Bit* 8-bit mode

### 29.6.14 enum usdhc\_endian\_mode\_t

Enumerator

*kUSDHC\_EndianModeBig* Big endian mode.  
*kUSDHC\_EndianModeHalfWordBig* Half word big endian mode.  
*kUSDHC\_EndianModeLittle* Little endian mode.

**29.6.15 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

**29.6.16 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.

**29.6.17 enum usdhc\_boot\_mode\_t**

Enumerator

*kUSDHC\_BootModeNormal* Normal boot.  
*kUSDHC\_BootModeAlternative* Alternative boot.

**29.6.18 enum usdhc\_card\_command\_type\_t**

Enumerator

*kCARD\_CommandTypeNormal* Normal command.  
*kCARD\_CommandTypeSuspend* Suspend command.  
*kCARD\_CommandTypeResume* Resume command.  
*kCARD\_CommandTypeAbort* Abort command.

**29.6.19 enum usdhc\_card\_response\_type\_t**

Define the command response type from card to host controller.

Enumerator

*kCARD\_ResponseNone* Response type: none.

## Enumeration Type Documentation

*kCARD\_ResponseTypeDefs*

- 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.

### 29.6.20 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.

### 29.6.21 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.

### 29.6.22 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

## 29.7 Function Documentation

### 29.7.1 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. |
|------------------------|-----------------------|

### 29.7.2 void USDHC\_Deinit ( ***USDHC\_Type*** \* *base* )

Parameters

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

### 29.7.3 bool USDHC\_Reset ( ***USDHC\_Type*** \* *base*, ***uint32\_t mask***, ***uint32\_t timeout*** )

Parameters

## Function Documentation

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | USDHC peripheral base address.     |
| <i>mask</i>    | The reset type mask(_usdhc_reset). |
| <i>timeout</i> | Timeout for reset.                 |

Return values

|              |                     |
|--------------|---------------------|
| <i>true</i>  | Reset successfully. |
| <i>false</i> | Reset failed.       |

**29.7.4 `status_t USDHC_SetAdmaTableConfig ( USDHC_Type * base, usdhc_adma_config_t * dmaConfig, usdhc_data_t * dataConfig, uint32_t flags )`**

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | USDHC peripheral base address. |
| <i>adma</i>    | configuration                  |
| <i>data</i>    | Data descriptor                |
| <i>command</i> | flags                          |

Return values

|                           |                                                             |
|---------------------------|-------------------------------------------------------------|
| <i>kStatus_OutOfRange</i> | ADMA descriptor table length isn't enough to describe data. |
| <i>kStatus_Success</i>    | Operate successfully.                                       |

**29.7.5 `static void USDHC_EnableInterruptStatus ( USDHC_Type * base, uint32_t mask ) [inline], [static]`**

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>base</i> | USDHC peripheral base address.                             |
| <i>mask</i> | Interrupt status flags mask(_usdhc_interrupt_status_flag). |

**29.7.6 `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). |

### 29.7.7 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). |

### 29.7.8 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). |

### 29.7.9 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).

### 29.7.10 static void USDHC\_ClearInterruptStatusFlags ( **USDHC\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

write 1 clears

## Function Documentation

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | USDHC peripheral base address.                                 |
| <i>mask</i> | The interrupt status flags mask(_usdhc_interrupt_status_flag). |

**29.7.11 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).

**29.7.12 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).

**29.7.13 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).

#### 29.7.14 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. |

#### 29.7.15 static void **USDHC\_ForceClockOn** ( **USDHC\_Type** \* *base*, **bool** *enable* ) [**inline**], [**static**]

Parameters

|                       |                                |
|-----------------------|--------------------------------|
| <i>base</i>           | USDHC peripheral base address. |
| <i>enable/disable</i> | flag.                          |

#### 29.7.16 **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

### 29.7.17 **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.       |

### 29.7.18 static void USDHC AssertHardwareReset ( **USDHC\_Type** \* *base*, **bool** *high* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USDHC peripheral base address. |
| <i>high</i> | 1 or 0 level                   |

### 29.7.19 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.           |

### 29.7.20 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

**29.7.21 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.

### 29.7.22 void USDHC\_SendCommand ( **USDHC\_Type** \* *base*, **usdhc\_command\_t** \* *command* )

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | USDHC peripheral base address. |
| <i>command</i> | configuration                  |

### 29.7.23 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.        |

### 29.7.24 static void USDHC\_CardDetectByData3 ( **USDHC\_Type** \* *base*, **bool** *enable* ) [inline], [static]

Parameters

|                       |                                |
|-----------------------|--------------------------------|
| <i>base</i>           | USDHC peripheral base address. |
| <i>enable/disable</i> | flag                           |

### 29.7.25 static bool USDHC\_DetectCardInsert ( **USDHC\_Type** \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

**29.7.26 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.                       |

**29.7.27 static void USDHC\_SetContinueRequest ( **USDHC\_Type** \* *base* ) [inline], [static]**

Parameters

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

**29.7.28 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. |

**29.7.29 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). |

**29.7.30 static void UDSHC\_SelectVoltage ( **USDHC\_Type** \* *base*, **bool** *en18v* )  
[inline], [static]**

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USDHC peripheral base address. |
| <i>true</i> | 1.8V, false 3.0V               |

**29.7.31 static bool USDHC\_RequestTuningForSDR50 ( **USDHC\_Type** \* *base* )  
[inline], [static]**

Parameters

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

**29.7.32 static bool USDHC\_RequestReTuning ( **USDHC\_Type** \* *base* ) [inline],  
[static]**

Parameters

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

## Function Documentation

29.7.33 **static void USDHC\_EnableAutoTuning( USDHC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|                       |                                |
|-----------------------|--------------------------------|
| <i>base</i>           | USDHC peripheral base address. |
| <i>enable/disable</i> | flag                           |

#### 29.7.34 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                  |

#### 29.7.35 static void USDHC\_EnableDDRMode ( **USDHC\_Type** \* *base*, **bool** *enable*, **uint32\_t** *nibblePos* ) [inline], [static]

Parameters

|                       |                                |
|-----------------------|--------------------------------|
| <i>base</i>           | USDHC peripheral base address. |
| <i>enable/disable</i> | flag                           |
| <i>nibble</i>         | position                       |

#### 29.7.36 void USDHC\_EnableAutoTuningForCmdAndData ( **USDHC\_Type** \* *base* )

Parameters

|                       |                                |
|-----------------------|--------------------------------|
| <i>base</i>           | USDHC peripheral base address. |
| <i>enable/disable</i> | flag                           |

the auto tuning enable for CMD/DATA line

Parameters

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

#### 29.7.37 void USDHC\_EnableManualTuning ( **USDHC\_Type** \* *base*, **bool** *enable* )

## Function Documentation

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USDHC peripheral base address. |
| <i>tuning</i> | enable flag                    |

**29.7.38 `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 |

**29.7.39 `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                           |
| <i>enable/disable</i> | flag                           |

**29.7.40 `static uint32_t USDHC_GetExecuteStdTuningStatus ( USDHC_Type * base ) [inline], [static]`**

Parameters

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

**29.7.41 static uint32\_t USDHC\_CheckStdTuningResult ( USDHC\_Type \* *base* )  
[inline], [static]**

Parameters

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

**29.7.42 static uint32\_t USDHC\_CheckTuningError ( USDHC\_Type \* *base* )  
[inline], [static]**

Parameters

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

**29.7.43 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.              |

## Function Documentation

Return values

|                                                   |                                 |
|---------------------------------------------------|---------------------------------|
| <i>kStatus_InvalidArgument</i>                    | Argument is invalid.            |
| <i>kStatus_USDHC_-PrepareAdmaDescriptorFailed</i> | Prepare ADMA descriptor failed. |
| <i>kStatus_USDHC_SendCommandFailed</i>            | Send command failed.            |
| <i>kStatus_USDHC_TransferDataFailed</i>           | Transfer data failed.           |
| <i>kStatus_Success</i>                            | Operate successfully.           |

**29.7.44 void USDHC\_TransferCreateHandle ( **USDHC\_Type** \* *base*,  
                 **usdhc\_handle\_t** \* *handle*, **const usdhc\_transfer\_callback\_t** \* *callback*,  
                 **void** \* *userData* )**

Parameters

|                 |                                                      |
|-----------------|------------------------------------------------------|
| <i>base</i>     | USDHC peripheral base address.                       |
| <i>handle</i>   | USDHC handle pointer.                                |
| <i>callback</i> | Structure pointer to contain all callback functions. |
| <i>userData</i> | Callback function parameter.                         |

**29.7.45 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.           |

#### 29.7.46 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.                  |

## Function Documentation

# Chapter 30

## WDOG: Watchdog Timer Driver

### 30.1 Overview

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

### 30.2 Typical use case

```
wdog_config_t config;
WDOG_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableWindowMode = true;
config.windowValue = 0x1ffU;
WDOG_Init(wdog_base, &config);
```

## Data Structures

- struct `wdog_work_mode_t`  
*Defines WDOG work mode. [More...](#)*
- struct `wdog_config_t`  
*Describes WDOG configuration structure. [More...](#)*
- struct `wdog_test_config_t`  
*Describes WDOG test mode configuration structure. [More...](#)*

## Enumerations

- enum `wdog_clock_source_t` {  
  `kWDOG_LpoClockSource` = 0U,  
  `kWDOG_AlternateClockSource` = 1U }  
*Describes WDOG clock source.*
- enum `wdog_clock_prescaler_t` {  
  `kWDOG_ClockPrescalerDivide1` = 0x0U,  
  `kWDOG_ClockPrescalerDivide2` = 0x1U,  
  `kWDOG_ClockPrescalerDivide3` = 0x2U,  
  `kWDOG_ClockPrescalerDivide4` = 0x3U,  
  `kWDOG_ClockPrescalerDivide5` = 0x4U,  
  `kWDOG_ClockPrescalerDivide6` = 0x5U,  
  `kWDOG_ClockPrescalerDivide7` = 0x6U,  
  `kWDOG_ClockPrescalerDivide8` = 0x7U }  
*Describes the selection of the clock prescaler.*
- enum `wdog_test_mode_t` {  
  `kWDOG_QuickTest` = 0U,  
  `kWDOG_Bytetest` = 1U }  
*Describes WDOG test mode.*

## Typical use case

- enum `wdog_tested_byte_t` {  
  `kWDOG_TestByte0` = 0U,  
  `kWDOG_TestByte1` = 1U,  
  `kWDOG_TestByte2` = 2U,  
  `kWDOG_TestByte3` = 3U }  
    *Describes WDOG tested byte selection in byte test mode.*
- enum `_wdog_interrupt_enable_t` { `kWDOG_InterruptEnable` = `WDOG_STCTRLH_IRQRSTEN_MASK` }  
    *WDOG interrupt configuration structure, default settings all disabled.*
- enum `_wdog_status_flags_t` {  
  `kWDOG_RunningFlag` = `WDOG_STCTRLH_WDOGEN_MASK`,  
  `kWDOG_TimeoutFlag` = `WDOG_STCTRLL_INTFLG_MASK` }  
    *WDOG status flags.*

## Driver version

- #define `FSL_WDOG_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
    *Defines WDOG driver version 2.0.0.*

## Unlock sequence

- #define `WDOG_FIRST_WORD_OF_UNLOCK` (0xC520U)  
    *First word of unlock sequence.*
- #define `WDOG_SECOND_WORD_OF_UNLOCK` (0xD928U)  
    *Second word of unlock sequence.*

## Refresh sequence

- #define `WDOG_FIRST_WORD_OF_REFRESH` (0xA602U)  
    *First word of refresh sequence.*
- #define `WDOG_SECOND_WORD_OF_REFRESH` (0xB480U)  
    *Second word of refresh sequence.*

## WDOG Initialization and De-initialization

- void `WDOG_GetDefaultConfig` (`wdog_config_t` \*config)  
    *Initializes 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)

- static void [WDOG\\_EnableInterrupts](#) (WDOG\_Type \*base, uint32\_t mask)
 

*Disables the WDOG module.*
- static void [WDOG\\_DisableInterrupts](#) (WDOG\_Type \*base, uint32\_t mask)
 

*Enables 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.*

## 30.3 Data Structure Documentation

### 30.3.1 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.*

### 30.3.2 struct wdog\_config\_t

#### Data Fields

- bool [enableWdog](#)

*Enables or disables WDOG.*
- [wdog\\_clock\\_source\\_t clockSource](#)

*Clock source select.*
- [wdog\\_clock\\_prescaler\\_t prescaler](#)

*Clock prescaler value.*
- [wdog\\_work\\_mode\\_t workMode](#)

*Configures WDOG work mode in debug stop and wait mode.*
- bool [enableUpdate](#)

*Update write-once register enable.*
- bool [enableInterrupt](#)

## Enumeration Type Documentation

- `bool enableWindowMode`  
*Enables or disables WDOG interrupt.*
- `uint32_t windowValue`  
*Enables or disables WDOG window mode.*
- `uint32_t timeoutValue`  
*Window value.*
- `uint32_t timeoutValue`  
*Timeout value.*

### 30.3.3 `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.*

## 30.4 Macro Definition Documentation

### 30.4.1 `#define FSL_WDOG_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

## 30.5 Enumeration Type Documentation

### 30.5.1 `enum wdog_clock_source_t`

Enumerator

`kWDOG_LpoClockSource` WDOG clock sourced from LPO.

`kWDOG_AlternateClockSource` WDOG clock sourced from alternate clock source.

### 30.5.2 `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.

### 30.5.3 enum wdog\_test\_mode\_t

Enumerator

*kWDOG\_QuickTest* Selects quick test.

*kWDOG\_ByeTest* Selects byte test.

### 30.5.4 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.

### 30.5.5 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.

### 30.5.6 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.

## 30.6 Function Documentation

### 30.6.1 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_IpoClockSource;
* wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
* wdogConfig->workMode.enableWait = true;
* wdogConfig->workMode.enableStop = false;
* wdogConfig->workMode.enableDebug = false;
* wdogConfig->enableUpdate = true;
* wdogConfig->enableInterrupt = false;
* wdogConfig->enableWindowMode = false;
* wdogConfig->windowValue = 0;
* wdogConfig->timeoutValue = 0xFFFFU;
*
```

### Parameters

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

### See Also

[wdog\\_config\\_t](#)

### 30.6.2 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    |

### 30.6.3 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.

### 30.6.4 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 = 0xffffffffu;
*     WDOG_SetTestModeConfig(wdog_base, &test_config);
*
```

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | WDOG peripheral base address              |
| <i>config</i> | The functional test configuration of WDOG |

### 30.6.5 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 |
|-------------|------------------------------|

### 30.6.6 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 |
|-------------|------------------------------|

### 30.6.7 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> |

### 30.6.8 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> |

### 30.6.9 **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.

**30.6.10 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 |

**30.6.11 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. |

### 30.6.12 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.           |

### 30.6.13 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 |
|-------------|------------------------------|

### 30.6.14 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 |
|-------------|------------------------------|

**30.6.15 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.

### 30.6.16 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 31

## Clock Driver

### 31.1 Overview

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

### 31.2 Get frequency

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

### 31.3 External clock frequency

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

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

## Data Structures

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

## Macros

- #define `FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL` 0

## External clock frequency

- `#define ADC_CLOCKS`  
*Clock ip name array for ADC.*
- `#define ADC_5HC_CLOCKS`  
*Clock ip name array for ADC\_5HC.*
- `#define ECSPI_CLOCKS`  
*Clock ip name array for ECSPI.*
- `#define ENET_CLOCKS`  
*Clock ip name array for ENET.*
- `#define EPIT_CLOCKS`  
*Clock ip name array for EPIT.*
- `#define FLEXCAN_CLOCKS`  
*Clock ip name array for FLEXCAN.*
- `#define FLEXCAN_PERIPH_CLOCKS`  
*Serial Clock ip name array for FLEXCAN.*
- `#define GPIO_CLOCKS`  
*Clock ip name array for GPIO.*
- `#define GPT_CLOCKS`  
*Clock ip name array for GPT.*
- `#define GPT_PERIPH_CLOCKS`  
*Serial Clock ip name array for GPT.*
- `#define I2C_CLOCKS`  
*Clock ip name array for I2C.*
- `#define PWM_CLOCKS`  
*Clock ip name array for PWM.*
- `#define QSPI_CLOCKS`  
*Clock ip name array for QSPI.*
- `#define SAI_CLOCKS`  
*Clock ip name array for SAI.*
- `#define SDMA_CLOCKS`  
*Clock ip name array for SDMA.*
- `#define SIM_CLOCKS`  
*Clock ip name array for SIM.*
- `#define TSC_CLOCKS`  
*Clock ip name array for TSC.*
- `#define UART_CLOCKS`  
*Clock ip name array for UART.*
- `#define USDHC_CLOCKS`  
*Clock ip name array for USDHC.*
- `#define WDOG_CLOCKS`  
*Clock ip name array for WDOG.*
- `#define LCDIF_CLOCKS`  
*eLCDIF apb\_clk.*
- `#define LCDIF_PERIPH_CLOCKS`  
*eLCDIF pix\_clk.*
- `#define PXP_CLOCKS`  
*PXP clock.*
- `#define SNVS_HP_CLOCKS`  
*Clock ip name array for SNVS HP.*
- `#define SNVS_LP_CLOCKS`  
*Clock ip name array for SNVS LP.*

- #define **CSI\_CLOCKS**  
*CSI clock.*
- #define **CSI\_MCLK\_CLOCKS**  
*CSI MCLK.*
- #define **FSL\_CLOCK\_MMDC\_IPG\_GATE\_COUNT** 2U  
*MMDC IPG clock.*
- #define **MMDC\_ACLK\_CLOCKS**  
*MMDC ACLK.*
- #define **kCLOCK\_CoreSysClk** **kCLOCK\_CpuClk**  
*For compatible with other platforms without CCM.*
- #define **CLOCK\_GetCoreSysClkFreq** **CLOCK\_GetCpuClkFreq**  
*For compatible with other platforms without CCM.*

## Enumerations

- enum **clock\_name\_t** {
 **kCLOCK\_CpuClk** = 0x0U,
 **kCLOCK\_AxiClk** = 0x1U,
 **kCLOCK\_AhbClk** = 0x2U,
 **kCLOCK\_IpgClk** = 0x3U,
 **kCLOCK\_MmdcClk** = 0x4U,
 **kCLOCK\_OscClk** = 0x5U,
 **kCLOCK\_RtcClk** = 0x6U,
 **kCLOCK\_ArmPllClk** = 0x7U,
 **kCLOCK\_Usb1PllClk** = 0x8U,
 **kCLOCK\_Usb1PllPfd0Clk** = 0x9U,
 **kCLOCK\_Usb1PllPfd1Clk** = 0xAU,
 **kCLOCK\_Usb1PllPfd2Clk** = 0xBU,
 **kCLOCK\_Usb1PllPfd3Clk** = 0xCU,
 **kCLOCK\_Usb2PllClk** = 0xDU,
 **kCLOCK\_SysPllClk** = 0xEU,
 **kCLOCK\_SysPllPfd0Clk** = 0xFU,
 **kCLOCK\_SysPllPfd1Clk** = 0x10U,
 **kCLOCK\_SysPllPfd2Clk** = 0x11U,
 **kCLOCK\_SysPllPfd3Clk** = 0x12U,
 **kCLOCK\_EnetPll0Clk** = 0x13U,
 **kCLOCK\_EnetPll1Clk** = 0x14U,
 **kCLOCK\_EnetPll2Clk** = 0x15U,
 **kCLOCK\_AudioPllClk** = 0x16U,
 **kCLOCK\_VideoPllClk** = 0x17U
 }
   
*Clock name used to get clock frequency.*
- enum **clock\_ip\_name\_t** ,

## External clock frequency

```
kCLOCK_AipsTz1 = (0U << 8) | 0x0U,
kCLOCK_AipsTz2 = (0U << 8) | 0x1U,
kCLOCK_Apbhdma = (0U << 8) | 0x2U,
kCLOCK_Asrc = (0U << 8) | 0x3U,
kCLOCK_Caamsm = (0U << 8) | 0x4U,
kCLOCK_Caamwa = (0U << 8) | 0x5U,
kCLOCK_Caamwi = (0U << 8) | 0x6U,
kCLOCK_Can1 = (0U << 8) | 0x7U,
kCLOCK_Can1S = (0U << 8) | 0x8U,
kCLOCK_Can2 = (0U << 8) | 0x9U,
kCLOCK_Can2S = (0U << 8) | 0xAU,
kCLOCK_CpuDbg = (0U << 8) | 0xBU,
kCLOCK_Gpt2 = (0U << 8) | 0xCU,
kCLOCK_Gpt2S = (0U << 8) | 0xDU,
kCLOCK_Uart2 = (0U << 8) | 0xEU,
kCLOCK_Gpio2 = (0U << 8) | 0xFU,
kCLOCK_Ecspi1 = (1U << 8) | 0x0U,
kCLOCK_Ecspi2 = (1U << 8) | 0x1U,
kCLOCK_Ecspi3 = (1U << 8) | 0x2U,
kCLOCK_Ecspi4 = (1U << 8) | 0x3U,
kCLOCK_Adc_5hc = (1U << 8) | 0x4U,
kCLOCK_Uart3 = (1U << 8) | 0x5U,
kCLOCK_Epit1 = (1U << 8) | 0x6U,
kCLOCK_Epit2 = (1U << 8) | 0x7U,
kCLOCK_Adc1 = (1U << 8) | 0x8U,
kCLOCK_SimS = (1U << 8) | 0x9U,
kCLOCK_Gpt1 = (1U << 8) | 0xAU,
kCLOCK_Gpt1S = (1U << 8) | 0xBU,
kCLOCK_Uart4 = (1U << 8) | 0xCU,
kCLOCK_Gpio1 = (1U << 8) | 0xDU,
kCLOCK_Csu = (1U << 8) | 0xEU,
kCLOCK_Gpio5 = (1U << 8) | 0xFU,
kCLOCK_Csi = (2U << 8) | 0x1U,
kCLOCK_I2c1S = (2U << 8) | 0x3U,
kCLOCK_I2c2S = (2U << 8) | 0x4U,
kCLOCK_I2c3S = (2U << 8) | 0x5U,
kCLOCK_Ocotp = (2U << 8) | 0x6U,
kCLOCK_IomuxcIpt = (2U << 8) | 0x7U,
kCLOCK_Ipmux1 = (2U << 8) | 0x8U,
kCLOCK_Ipmux2 = (2U << 8) | 0x9U,
kCLOCK_Ipmux3 = (2U << 8) | 0xAU,
kCLOCK_Ipsync = (2U << 8) | 0xBU,
kCLOCK_Gpio3 = (2U << 8) | 0xDU,
kCLOCK_Lcd = (2U << 8) | 0xEU,
kCLOCK_Pxp = (2U << 8) | 0xFU,
kCLOCK_CsiMclk = (3U << 8) | 0x0U,
kCLOCK_Uart5 = (3U << 8) | 0x1U
```

kCLOCK\_Uart5 = (3U << 8) | 0x1U **MCC8Xpresso SDK API Reference Manual**

- ```

kCLOCK_Pwm7 = (6U << 8) | 0xFU }

Clock name used to enable/disable gate.
• enum clock_osc_t {
    kCLOCK_RcOsc = 0U,
    kCLOCK_XtalOsc = 1U }

OSC 24M sorce select.
• enum clock_gate_value_t {
    kCLOCK_ClockNotNeeded = 0U,
    kCLOCK_ClockNeededRun = 1U,
    kCLOCK_ClockNeededRunWait = 3U }

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

System clock mode.
• enum clock_mux_t {
    kCLOCK_StepMux = CCM_TUPLE(CCSR, CCM_CCSR_STEP_SEL_SHIFT, CCM_CCSR_ST-
EP_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_SecMux = CCM_TUPLE(CCSR, CCM_CCSR_SECONDARY_CLK_SEL_SHIFT, C-
CM_CCSR_SECONDARY_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_Pll1SwMux = CCM_TUPLE(CCSR, CCM_CCSR_PLL1_SW_CLK_SEL_SHIFT, CC-
M_CCSR_PLL1_SW_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_Pll3SwMux = CCM_TUPLE(CCSR, CCM_CCSR_PLL3_SW_CLK_SEL_SHIFT, CC-
M_CCSR_PLL3_SW_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_Periph2Mux = CCM_TUPLE(CBCDR, CCM_CBCDR_PERIPH2_CLK_SEL_SHIFT, CC-
M_CBCDR_PERIPH2_CLK_SEL_MASK, CCM_CDHIPR_PERIPH2_CLK_SEL_BUSY_S-
HIFT),
    kCLOCK_PeriphMux = CCM_TUPLE(CBCDR, CCM_CBCDR_PERIPH_CLK_SEL_SHIFT, CC-
M_CBCDR_PERIPH_CLK_SEL_MASK, CCM_CDHIPR_PERIPH_CLK_SEL_BUSY_SHI-
FT),
    kCLOCK_AxiAltMux = CCM_TUPLE(CBCDR, CCM_CBCDR_AXI_ALT_CLK_SEL_SHIFT, CC-
M_CBCDR_AXI_ALT_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_AxiMux = CCM_TUPLE(CBCDR, CCM_CBCDR_AXI_CLK_SEL_SHIFT, CCM_C-
BCDR_AXI_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_PrePeriph2Mux = CCM_TUPLE(CBCMR, CCM_CBCMR_PRE_PERIPH2_CLK_SE-
L_SHIFT, CCM_CBCMR_PRE_PERIPH2_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_PrePeriphMux = CCM_TUPLE(CBCMR, CCM_CBCMR_PRE_PERIPH_CLK_SEL_-_
SHIFT, CCM_CBCMR_PRE_PERIPH_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_Periph2Clk2Mux = CCM_TUPLE(CBCMR, CCM_CBCMR_PERIPH2_CLK2_SEL_-_
SHIFT, CCM_CBCMR_PERIPH2_CLK2_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_PeriphClk2Mux = CCM_TUPLE(CBCMR, CCM_CBCMR_PERIPH_CLK2_SEL_SHI-
FT, CCM_CBCMR_PERIPH_CLK2_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_EimSlowMux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_ACLK_EIM_SLOW_SE-
L_SHIFT, CCM_CSCMR1_ACLK_EIM_SLOW_SEL_MASK, CCM_NO_BUSY_WAIT),
    kCLOCK_GpmiMux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_GPMI_CLK_SEL_SHIFT, C-
}

```

## External clock frequency

```
CM_CSCMR1_GPMI_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_BchMux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_BCH_CLK_SEL_SHIFT, CC-
M_CSCMR1_BCH_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Usdhc2Mux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_USDHC2_CLK_SEL_SHI-
FT, CCM_CSCMR1_USDHC2_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Usdhc1Mux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_USDHC1_CLK_SEL_SHI-
FT, CCM_CSCMR1_USDHC1_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Sai3Mux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_SAI3_CLK_SEL_SHIFT, CC-
M_CSCMR1_SAI3_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Sai2Mux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_SAI2_CLK_SEL_SHIFT, CC-
M_CSCMR1_SAI2_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Sai1Mux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_SAI1_CLK_SEL_SHIFT, CC-
M_CSCMR1_SAI1_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Qspi1Mux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_QSPI1_CLK_SEL_SHIFT, C-
CM_CSCMR1_QSPI1_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_PerclkMux = CCM_TUPLE(CSCMR1, CCM_CSCMR1_PERCLK_CLK_SEL_SHIFT,
CCM_CSCMR1_PERCLK_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_VidMux = CCM_TUPLE(CSCMR2, CCM_CSCMR2_VID_CLK_SEL_SHIFT, CCM-
_CSCMR2_VID_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_CanMux = CCM_TUPLE(CSCMR2, CCM_CSCMR2_CAN_CLK_SEL_SHIFT, CC-
M_CSCMR2_CAN_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_UartMux = CCM_TUPLE(CSCDR1, CCM_CSCDR1_UART_CLK_SEL_SHIFT, CC-
M_CSCDR1_UART_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_EnfcMux = CCM_TUPLE(CS2CDR, CCM_CS2CDR_ENFC_CLK_SEL_SHIFT, CC-
M_CS2CDR_ENFC_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_LdbDi0Mux = CCM_TUPLE(CS2CDR, CCM_CS2CDR_LDB_DI0_CLK_SEL_SHIF-
T, CCM_CS2CDR_LDB_DI0_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_SpdifMux = CCM_TUPLE(CDCDR, CCM_CDCDR_SPDIF0_CLK_SEL_SHIFT, CC-
M_CDCDR_SPDIF0_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_SimPreMux = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_SIM_PRE_CLK_SEL_-
SHIFT, CCM_CHSCCDR_SIM_PRE_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_SimMux = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_SIM_CLK_SEL_SHIFT, C-
CM_CHSCCDR_SIM_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_EcspiMux = CCM_TUPLE(CSCDR2, CCM_CSCDR2_ECSPI_CLK_SEL_SHIFT, C-
CM_CSCDR2_ECSPI_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Lcdif1PreMux = CCM_TUPLE(CSCDR2, CCM_CSCDR2_LCDIF1_PRE_CLK_SEL-
_SHIFT, CCM_CSCDR2_LCDIF1_PRE_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_Lcdif1Mux = CCM_TUPLE(CSCDR2, CCM_CSCDR2_LCDIF1_CLK_SEL_SHIFT,
CCM_CSCDR2_LCDIF1_CLK_SEL_MASK, CCM_NO_BUSY_WAIT),
kCLOCK_CsiMux = CCM_TUPLE(CSCDR3, CCM_CSCDR3_CSI_CLK_SEL_SHIFT, CCM-
CSCDR3_CSI_CLK_SEL_MASK, CCM_NO_BUSY_WAIT) }
```

*MUX control names for clock mux setting.*

- enum `clock_div_t`{  
    kCLOCK\_ArmDiv = CCM\_TUPLE(CACRR, CCM\_CACRR\_ARM\_PODF\_SHIFT, CCM\_CAC-

RR\_ARM\_PODF\_MASK, CCM\_CDHIPR\_ARM\_PODF\_BUSY\_SHIFT),  
**kCLOCK\_PeriphClk2Div** = CCM\_TUPLE(CBCDR, CCM\_CBCDR\_PERIPH\_CLK2\_PODF\_SHIFT, CCM\_CBCDR\_PERIPH\_CLK2\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_Periph2Clk2Div** = CCM\_TUPLE(CBCDR, CCM\_CBCDR\_PERIPH2\_CLK2\_PODF\_SHIFT, CCM\_CBCDR\_PERIPH2\_CLK2\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_AxiDiv** = CCM\_TUPLE(CBCDR, CCM\_CBCDR\_AXI\_PODF\_SHIFT, CCM\_CBCDR\_AXI\_PODF\_MASK, CCM\_CDHIPR\_AXI\_PODF\_BUSY\_SHIFT),  
**kCLOCK\_AhbDiv** = CCM\_TUPLE(CBCDR, CCM\_CBCDR\_AHB\_PODF\_SHIFT, CCM\_CBCDR\_AHB\_PODF\_MASK, CCM\_CDHIPR\_AHB\_PODF\_BUSY\_SHIFT),  
**kCLOCK\_IpgDiv** = CCM\_TUPLE(CBCDR, CCM\_CBCDR\_IPG\_PODF\_SHIFT, CCM\_CBCDR\_IPG\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_FabricMmdcDiv** = CCM\_TUPLE(CBCDR, CCM\_CBCDR\_FABRIC\_MMDC\_PODF\_SHIFT, CCM\_CBCDR\_FABRIC\_MMDC\_PODF\_MASK, CCM\_CDHIPR\_MMDC\_PODF\_BUSY\_SHIFT),  
**kCLOCK\_Lcdif1Div** = CCM\_TUPLE(CBCMR, CCM\_CBCMR\_LCDIF1\_PODF\_SHIFT, CCM\_CBCMR\_LCDIF1\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_Qspi1Div** = CCM\_TUPLE(CSCMR1, CCM\_CSCMR1\_QSPI1\_PODF\_SHIFT, CCM\_CSCMR1\_QSPI1\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_EimSlowDiv** = CCM\_TUPLE(CSCMR1, CCM\_CSCMR1\_ACLK\_EIM\_SLOW\_PODF\_SHIFT, CCM\_CSCMR1\_ACLK\_EIM\_SLOW\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_PerclkDiv** = CCM\_TUPLE(CSCMR1, CCM\_CSCMR1\_PERCLK\_PODF\_SHIFT, CCM\_CSCMR1\_PERCLK\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_VidDiv** = CCM\_TUPLE(CSCMR2, CCM\_CSCMR2\_VID\_CLK\_PODF\_SHIFT, CCM\_CSCMR2\_VID\_CLK\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_VidPreDiv** = CCM\_TUPLE(CSCMR2, CCM\_CSCMR2\_VID\_CLK\_PRE\_PODF\_SHIFT, CCM\_CSCMR2\_VID\_CLK\_PRE\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_LdbDi0Div** = CCM\_TUPLE(CSCMR2, CCM\_CSCMR2\_LDB\_DI0\_DIV\_SHIFT, CCM\_CSCMR2\_LDB\_DI0\_DIV\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_LdbDi1Div** = CCM\_TUPLE(CSCMR2, CCM\_CSCMR2\_LDB\_DI1\_DIV\_SHIFT, CCM\_CSCMR2\_LDB\_DI1\_DIV\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_CanDiv** = CCM\_TUPLE(CSCMR2, CCM\_CSCMR2\_CAN\_CLK\_PODF\_SHIFT, CCM\_CSCMR2\_CAN\_CLK\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_GpmiDiv** = CCM\_TUPLE(CSCDR1, CCM\_CSCDR1\_GPMI\_PODF\_SHIFT, CCM\_CSCDR1\_GPMI\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_BchDiv** = CCM\_TUPLE(CSCDR1, CCM\_CSCDR1\_BCH\_PODF\_SHIFT, CCM\_CSCDR1\_BCH\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_Usdhc2Div** = CCM\_TUPLE(CSCDR1, CCM\_CSCDR1\_USDHC2\_PODF\_SHIFT, CCM\_CSCDR1\_USDHC2\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_Usdhc1Div** = CCM\_TUPLE(CSCDR1, CCM\_CSCDR1\_USDHC1\_PODF\_SHIFT, CCM\_CSCDR1\_USDHC1\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_UartDiv** = CCM\_TUPLE(CSCDR1, CCM\_CSCDR1\_UART\_CLK\_PODF\_SHIFT, CCM\_CSCDR1\_UART\_CLK\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_Sai3PreDiv** = CCM\_TUPLE(CS1CDR, CCM\_CS1CDR\_SAI3\_CLK\_PRED\_SHIFT, CCM\_CS1CDR\_SAI3\_CLK\_PRED\_MASK, CCM\_NO\_BUSY\_WAIT),  
**kCLOCK\_Sai3Div** = CCM\_TUPLE(CS1CDR, CCM\_CS1CDR\_SAI3\_CLK\_PODF\_SHIFT, CCM\_CS1CDR\_SAI3\_CLK\_PODF\_MASK, CCM\_NO\_BUSY\_WAIT),

## External clock frequency

```
M_CS1CDR_SAI3_CLK_PODF_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_Sai1PreDiv = CCM_TUPLE(CS1CDR, CCM_CS1CDR_SAI1_CLK_PRED_SHIFT, C-  
CM_CS1CDR_SAI1_CLK_PRED_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_Sai1Div = CCM_TUPLE(CS1CDR, CCM_CS1CDR_SAI1_CLK_PODF_SHIFT, CC-  
M_CS1CDR_SAI1_CLK_PODF_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_EnfcPreDiv = CCM_TUPLE(CS2CDR, CCM_CS2CDR_ENFC_CLK_PRED_SHIFT,  
CCM_CS2CDR_ENFC_CLK_PRED_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_EnfcDiv = CCM_TUPLE(CS2CDR, CCM_CS2CDR_ENFC_CLK_PODF_SHIFT, CC-  
M_CS2CDR_ENFC_CLK_PODF_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_Sai2PreDiv = CCM_TUPLE(CS2CDR, CCM_CS2CDR_SAI2_CLK_PRED_SHIFT, C-  
CM_CS2CDR_SAI2_CLK_PRED_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_Sai2Div = CCM_TUPLE(CS2CDR, CCM_CS2CDR_SAI2_CLK_PODF_SHIFT, CC-  
M_CS2CDR_SAI2_CLK_PODF_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_Spdif0PreDiv = CCM_TUPLE(CCDCDR, CCM_CDCCDR_SPDIF0_CLK_PRED_SHIF-  
T, CCM_CDCCDR_SPDIF0_CLK_PRED_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_Spdif0Div = CCM_TUPLE(CCDCDR, CCM_CDCCDR_SPDIF0_CLK_PODF_SHIFT, C-  
CM_CDCCDR_SPDIF0_CLK_PODF_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_SimDiv = CCM_TUPLE(CHSCCDR, CCM_CHSCCDR_SIM_PODF_SHIFT, CCM_-  
CHSCCDR_SIM_PODF_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_EcspiDiv = CCM_TUPLE(CSCDR2, CCM_CSCDR2_ECSPI_CLK_PODF_SHIFT, C-  
CM_CSCDR2_ECSPI_CLK_PODF_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_Lcdif1PreDiv = CCM_TUPLE(CSCDR2, CCM_CSCDR2_LCDIF1_PRED_SHIFT, C-  
CM_CSCDR2_LCDIF1_PRED_MASK, CCM_NO_BUSY_WAIT),  
kCLOCK_CsiDiv = CCM_TUPLE(CSCDR3, CCM_CSCDR3_CSI_PODF_SHIFT, CCM_CSCD-  
R3_CSI_PODF_MASK, CCM_NO_BUSY_WAIT) }
```

*DIV control names for clock div setting.*

- enum `clock_pll_t`{  
kCLOCK\_PlIArm = 0U,  
kCLOCK\_PlISys = 1U,  
kCLOCK\_PlIUsb1 = 2U,  
kCLOCK\_PlIAudio = 3U,  
kCLOCK\_PlIVideo = 4U,  
kCLOCK\_PlIEnet0 = 5U,  
kCLOCK\_PlIEnet1 = 6U,  
kCLOCK\_PlIEnet2 = 7U,  
kCLOCK\_PlIUsb2 = 8U }

*PLL name.*

- enum `clock_pfd_t`{  
kCLOCK\_Pfd0 = 0U,  
kCLOCK\_Pfd1 = 1U,  
kCLOCK\_Pfd2 = 2U,  
kCLOCK\_Pfd3 = 3U }

*PLL PFD name.*

- enum `clock_usb_src_t`{  
kCLOCK\_Usb480M = 0,

- `kCLOCK_UsbSrcUnused = 0xFFFFFFFFU }`  
*USB clock source definition.*
- enum `clock_usb_phy_src_t` { `kCLOCK_Usbphy480M = 0` }  
*Source of the USB HS PHY.*

## Functions

- static void `CLOCK_SetMux (clock_mux_t mux, uint32_t value)`  
*Set CCM MUX node to certain value.*
- static uint32\_t `CLOCK_GetMux (clock_mux_t mux)`  
*Get CCM MUX value.*
- static void `CLOCK_SetDiv (clock_div_t divider, uint32_t value)`  
*Set CCM DIV node to certain value.*
- static uint32\_t `CLOCK_GetDiv (clock_div_t divider)`  
*Get CCM DIV node value.*
- static void `CLOCK_ControlGate (clock_ip_name_t name, clock_gate_value_t value)`  
*Control the clock gate for specific IP.*
- static void `CLOCK_EnableClock (clock_ip_name_t name)`  
*Enable the clock for specific IP.*
- static void `CLOCK_DisableClock (clock_ip_name_t name)`  
*Disable the clock for specific IP.*
- static void `CLOCK_SetMode (clock_mode_t mode)`  
*Setting the low power mode that system will enter on next assertion of dsm\_request signal.*
- uint32\_t `CLOCK_GetFreq (clock_name_t name)`  
*Gets the clock frequency for a specific clock name.*

## Variables

- uint32\_t `g_xtalFreq`  
*External XTAL (24M OSC/SYSOSC) clock frequency.*
- uint32\_t `g_rtcXtalFreq`  
*External RTC XTAL (32K OSC) clock frequency.*

## Driver version

- #define `FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`  
*CLOCK driver version 2.1.0.*

## OSC operations

- void `CLOCK_InitExternalClk (bool bypassXtalOsc)`  
*Initialize the external 24MHz clock.*
- void `CLOCK_DeinitExternalClk (void)`  
*Deinitialize the external 24MHz clock.*
- void `CLOCK_SwitchOsc (clock_osc_t osc)`  
*Switch the OSC.*
- static uint32\_t `CLOCK_GetOscFreq (void)`  
*Gets the OSC clock frequency.*
- static uint32\_t `CLOCK_GetRtcFreq (void)`  
*Gets the RTC clock frequency.*

## External clock frequency

- static void **CLOCK\_SetXtalFreq** (uint32\_t freq)  
*Set the XTAL (24M OSC) frequency based on board setting.*
- static void **CLOCK\_SetRtcXtalFreq** (uint32\_t freq)  
*Set the RTC XTAL (32K OSC) frequency based on board setting.*
- void **CLOCK\_InitRcOsc24M** (void)  
*Initialize the RC oscillator 24MHz clock.*
- void **CLOCK\_DeinitRcOsc24M** (void)  
*Power down the RCOSC 24M clock.*

## PLL/PFD operations

- void **CLOCK\_InitArmPll** (const **clock\_arm\_pll\_config\_t** \*config)  
*Initialize the ARM PLL.*
- void **CLOCK\_DeinitArmPll** (void)  
*De-initialize the ARM PLL.*
- void **CLOCK\_InitSysPll** (const **clock\_sys\_pll\_config\_t** \*config)  
*Initialize the System PLL.*
- void **CLOCK\_DeinitSysPll** (void)  
*De-initialize the System PLL.*
- void **CLOCK\_InitUsb1Pll** (const **clock\_usb\_pll\_config\_t** \*config)  
*Initialize the USB1 PLL.*
- void **CLOCK\_DeinitUsb1Pll** (void)  
*Deinitialize the USB1 PLL.*
- void **CLOCK\_InitUsb2Pll** (const **clock\_usb\_pll\_config\_t** \*config)  
*Initialize the USB2 PLL.*
- void **CLOCK\_DeinitUsb2Pll** (void)  
*Deinitialize the USB2 PLL.*
- void **CLOCK\_InitAudioPll** (const **clock\_audio\_pll\_config\_t** \*config)  
*Initializes the Audio PLL.*
- void **CLOCK\_DeinitAudioPll** (void)  
*De-initialize the Audio PLL.*
- void **CLOCK\_InitVideoPll** (const **clock\_video\_pll\_config\_t** \*config)  
*Initialize the video PLL.*
- void **CLOCK\_DeinitVideoPll** (void)  
*De-initialize the Video PLL.*
- void **CLOCK\_InitEnetPll** (const **clock\_enet\_pll\_config\_t** \*config)  
*Initialize the ENET PLL.*
- void **CLOCK\_DeinitEnetPll** (void)  
*Deinitialize the ENET PLL.*
- uint32\_t **CLOCK\_GetPllFreq** (**clock\_pll\_t** pll)  
*Get current PLL output frequency.*
- void **CLOCK\_InitSysPfd** (**clock\_pfd\_t** pfd, uint8\_t pfdFrac)  
*Initialize the System PLL PFD.*
- void **CLOCK\_DeinitSysPfd** (**clock\_pfd\_t** pfd)  
*De-initialize the System PLL PFD.*
- void **CLOCK\_InitUsb1Pfd** (**clock\_pfd\_t** pfd, uint8\_t pfdFrac)  
*Initialize the USB1 PLL PFD.*
- void **CLOCK\_DeinitUsb1Pfd** (**clock\_pfd\_t** pfd)  
*De-initialize the USB1 PLL PFD.*
- uint32\_t **CLOCK\_GetSysPfdFreq** (**clock\_pfd\_t** pfd)  
*Get current System PLL PFD output frequency.*

- `uint32_t CLOCK_GetUsb1PfdFreq (clock_pfd_t pfd)`  
*Get current USB1 PLL PFD output frequency.*
- `bool CLOCK_EnableUsbhs0Clock (clock_usb_src_t src, uint32_t freq)`  
*Enable USB HS clock.*
- `bool CLOCK_EnableUsbhs0PhyPllClock (clock_usb_phy_src_t src, uint32_t freq)`  
*Enable USB HS PHY PLL clock.*
- `void CLOCK_DisableUsbhs0PhyPllClock (void)`  
*Disable USB HS PHY PLL clock.*
- `bool CLOCK_EnableUsbhs1Clock (clock_usb_src_t src, uint32_t freq)`  
*Enable USB HS clock.*
- `bool CLOCK_EnableUsbhs1PhyPllClock (clock_usb_phy_src_t src, uint32_t freq)`  
*Enable USB HS PHY PLL clock.*
- `void CLOCK_DisableUsbhs1PhyPllClock (void)`  
*Disable USB HS PHY PLL clock.*

## 31.4 Data Structure Documentation

### 31.4.1 struct clock\_arm\_pll\_config\_t

#### Data Fields

- `uint32_t loopDivider`  
*PLL loop divider.*

#### 31.4.1.0.0.65 Field Documentation

##### 31.4.1.0.0.65.1 uint32\_t clock\_arm\_pll\_config\_t::loopDivider

Valid range for divider value: 54-108.  $F_{out} = F_{in} * \text{loopDivider} / 2$ .

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

#### Data Fields

- `uint8_t loopDivider`  
*PLL loop divider.*

#### 31.4.2.0.0.66 Field Documentation

##### 31.4.2.0.0.66.1 uint8\_t clock\_usb\_pll\_config\_t::loopDivider

0 -  $F_{out} = F_{ref} * 20$ ; 1 -  $F_{out} = F_{ref} * 22$

## Data Structure Documentation

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

#### Data Fields

- uint8\_t **loopDivider**  
*PLL loop divider.*
- uint32\_t **numerator**  
*30 bit numerator of fractional loop divider.*
- uint32\_t **denominator**  
*30 bit denominator of fractional loop divider*

#### 31.4.3.0.0.67 Field Documentation

##### 31.4.3.0.0.67.1 uint8\_t clock\_sys\_pll\_config\_t::loopDivider

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

##### 31.4.3.0.0.67.2 uint32\_t clock\_sys\_pll\_config\_t::numerator

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

#### Data Fields

- uint8\_t **loopDivider**  
*PLL loop divider.*
- uint8\_t **postDivider**  
*Divider after the PLL, should only be 1, 2, 4, 8, 16.*
- uint32\_t **numerator**  
*30 bit numerator of fractional loop divider.*
- uint32\_t **denominator**  
*30 bit denominator of fractional loop divider*

#### 31.4.4.0.0.68 Field Documentation

##### 31.4.4.0.0.68.1 uint8\_t clock\_audio\_pll\_config\_t::loopDivider

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

##### 31.4.4.0.0.68.2 uint8\_t clock\_audio\_pll\_config\_t::postDivider

##### 31.4.4.0.0.68.3 uint32\_t clock\_audio\_pll\_config\_t::numerator

### 31.4.5 struct clock\_video\_pll\_config\_t

#### Data Fields

- uint8\_t **loopDivider**  
*PLL loop divider.*

- **uint8\_t postDivider**  
*Divider after the PLL, should only be 1, 2, 4, 8, 16.*
- **uint32\_t numerator**  
*30 bit numerator of fractional loop divider.*
- **uint32\_t denominator**  
*30 bit denominator of fractional loop divider*

### 31.4.5.0.0.69 Field Documentation

#### 31.4.5.0.0.69.1 **uint8\_t clock\_video\_pll\_config\_t::loopDivider**

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

#### 31.4.5.0.0.69.2 **uint8\_t clock\_video\_pll\_config\_t::postDivider**

#### 31.4.5.0.0.69.3 **uint32\_t clock\_video\_pll\_config\_t::numerator**

### 31.4.6 **struct clock\_enet\_pll\_config\_t**

## Data Fields

- **bool enableClkOutput0**  
*Power on and enable PLL clock output for ENET0 (ref\_enetpll0).*
- **bool enableClkOutput1**  
*Power on and enable PLL clock output for ENET1 (ref\_enetpll1).*
- **bool enableClkOutput2**  
*Power on and enable PLL clock output for ENET2 (ref\_enetpll2).*
- **uint8\_t loopDivider0**  
*Controls the frequency of the ENET0 reference clock.*
- **uint8\_t loopDivider1**  
*Controls the frequency of the ENET1 reference clock.*

### 31.4.6.0.0.70 Field Documentation

#### 31.4.6.0.0.70.1 **bool clock\_enet\_pll\_config\_t::enableClkOutput0**

#### 31.4.6.0.0.70.2 **bool clock\_enet\_pll\_config\_t::enableClkOutput1**

#### 31.4.6.0.0.70.3 **bool clock\_enet\_pll\_config\_t::enableClkOutput2**

#### 31.4.6.0.0.70.4 **uint8\_t clock\_enet\_pll\_config\_t::loopDivider0**

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

#### 31.4.6.0.0.70.5 **uint8\_t clock\_enet\_pll\_config\_t::loopDivider1**

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

## Macro Definition Documentation

### 31.5 Macro Definition Documentation

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

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

Note

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

#### 31.5.2 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

#### 31.5.3 #define ADC\_CLOCKS

**Value:**

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

#### 31.5.4 #define ADC\_5HC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Adc_5hc           \
}
```

#### 31.5.5 #define ECSPi\_CLOCKS

**Value:**

```
{           \
    kCLOCK_IpInvalid,  kCLOCK_Ecspi1,  kCLOCK_Ecspi2,           \
    kCLOCK_Ecspi3,  kCLOCK_Ecspi4           \
}
```

**31.5.6 #define ENET\_CLOCKS****Value:**

```
{
    kCLOCK_Invalid, kCLOCK_Enet, kCLOCK_Enet
}
```

**31.5.7 #define EPIT\_CLOCKS****Value:**

```
{
    kCLOCK_Invalid, kCLOCK_Epit1, kCLOCK_Epit2
}
```

**31.5.8 #define FLEXCAN\_CLOCKS****Value:**

```
{
    kCLOCK_Invalid, kCLOCK_Can1, kCLOCK_Can2
}
```

**31.5.9 #define FLEXCAN\_PERIPH\_CLOCKS****Value:**

```
{
    kCLOCK_Invalid, kCLOCK_Can1S, kCLOCK_Can2S
}
```

**31.5.10 #define GPIO\_CLOCKS****Value:**

```
{
    kCLOCK_Invalid, kCLOCK_Gpio1, kCLOCK_Gpio2,
    \ kCLOCK_Gpio3, kCLOCK_Gpio4, kCLOCK_Gpio5
}
```

## Macro Definition Documentation

### 31.5.11 #define GPT\_CLOCKS

**Value:**

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

### 31.5.12 #define GPT\_PERIPH\_CLOCKS

**Value:**

```
{           \
    kCLOCK_IpInvalid, kCLOCK_Gpt1S, kCLOCK_Gpt2S           \
}
```

### 31.5.13 #define I2C\_CLOCKS

**Value:**

```
{           \
    kCLOCK_IpInvalid, kCLOCK_I2c1S, kCLOCK_I2c2S,           \
    \           \
    kCLOCK_I2c3S, kCLOCK_I2c4S           \
}
```

### 31.5.14 #define PWM\_CLOCKS

**Value:**

```
{           \
    kCLOCK_IpInvalid,           \
    kCLOCK_Pwm1, kCLOCK_Pwm2, kCLOCK_Pwm3,           \
    kCLOCK_Pwm4,           \
    kCLOCK_Pwm5, kCLOCK_Pwm6, kCLOCK_Pwm7,           \
    kCLOCK_Pwm8,           \
}
```

### 31.5.15 #define QSPI\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Qspil           \
}
```

**31.5.16 #define SAI\_CLOCKS****Value:**

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

**31.5.17 #define SDMA\_CLOCKS****Value:**

```
{
    kCLOCK_Sdma \
}
```

**31.5.18 #define SIM\_CLOCKS****Value:**

```
{
    kCLOCK_IpInvalid, kCLOCK_Sim1, kCLOCK_Sim2,
}
```

**31.5.19 #define TSC\_CLOCKS****Value:**

```
{
    kCLOCK_Tsc \
}
```

**31.5.20 #define UART\_CLOCKS****Value:**

```
{
    kCLOCK_IpInvalid,
    kCLOCK_Uart1, kCLOCK_Uart2, kCLOCK_Uart3,
    kCLOCK_Uart4, \
    kCLOCK_Uart5, kCLOCK_Uart6, kCLOCK_Uart7,
    kCLOCK_Uart8 \
}
```

## Macro Definition Documentation

### 31.5.21 #define USDHC\_CLOCKS

**Value:**

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

### 31.5.22 #define WDOG\_CLOCKS

**Value:**

```
{           \
    kCLOCK_IpInvalid, kCLOCK_Wdog1, kCLOCK_Wdog2,
    kCLOCK_Wdog3 \
}
```

### 31.5.23 #define LCDIF\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Lcd \
}
```

### 31.5.24 #define LCDIF\_PERIPH\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Lcdif1 \
}
```

### 31.5.25 #define PXP\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Pxp \
}
```

**31.5.26 #define SNVS\_HP\_CLOCKS****Value:**{  
 kCLOCK\_SnvsHp \\\br/>}**31.5.27 #define SNVS\_LP\_CLOCKS****Value:**{  
 kCLOCK\_SnvsLp \\\br/>}**31.5.28 #define CSI\_CLOCKS****Value:**{  
 kCLOCK\_Csi \\\br/>}**31.5.29 #define CSI\_MCLK\_CLOCKS****Value:**{  
 kCLOCK\_CsiMclk \\\br/>}**31.5.30 #define FSL\_CLOCK\_MMDC\_IPG\_GATE\_COUNT 2U****31.5.31 #define MMDC\_ACLK\_CLOCKS****Value:**{  
 kCLOCK\_MmdcAClk \\\br/>}

## Enumeration Type Documentation

31.5.32 `#define kCLOCK_CoreSysClk kCLOCK_CpuClk`

31.5.33 `#define CLOCK_GetCoreSysClkFreq CLOCK_GetCpuClkFreq`

## 31.6 Enumeration Type Documentation

### 31.6.1 `enum clock_name_t`

Enumerator

*kCLOCK\_CpuClk* CPU clock.

*kCLOCK\_AxiClk* AXI clock.

*kCLOCK\_AhbClk* AHB clock.

*kCLOCK\_IpgClk* IPG clock.

*kCLOCK\_MmdcClk* MMDC clock.

*kCLOCK\_OscClk* OSC clock selected by PMU\_LOWPWR\_CTRL[OSC\_SEL].

*kCLOCK\_RtcClk* RTC clock. (RTCCLK)

*kCLOCK\_ArmPllClk* ARMPLLCLK.

*kCLOCK\_Usb1PllClk* USB1PLLCLK.

*kCLOCK\_Usb1PllPfd0Clk* USB1PLLPFD0CLK.

*kCLOCK\_Usb1PllPfd1Clk* USB1PLLPFD1CLK.

*kCLOCK\_Usb1PllPfd2Clk* USB1PLLPFD2CLK.

*kCLOCK\_Usb1PllPfd3Clk* USB1PLLPFD3CLK.

*kCLOCK\_Usb2PllClk* USB2PLLCLK.

*kCLOCK\_SysPllClk* SYSPLLCLK.

*kCLOCK\_SysPllPfd0Clk* SYSPLLPFD0CLK.

*kCLOCK\_SysPllPfd1Clk* SYSPLLPFD1CLK.

*kCLOCK\_SysPllPfd2Clk* SYSPLLPFD2CLK.

*kCLOCK\_SysPllPfd3Clk* SYSPLLPFD3CLK.

*kCLOCK\_EnetPll0Clk* Enet PLLCLK ref\_enetpll0.

*kCLOCK\_EnetPll1Clk* Enet PLLCLK ref\_enetpll1.

*kCLOCK\_EnetPll2Clk* Enet PLLCLK ref\_enetpll2.

*kCLOCK\_AudioPllClk* Audio PLLCLK.

*kCLOCK\_VideoPllClk* Video PLLCLK.

### 31.6.2 `enum clock_ip_name_t`

Enumerator

*kCLOCK\_AipsTz1* CCGR0, CG0.

*kCLOCK\_AipsTz2* CCGR0, CG1.

*kCLOCK\_Apbhdma* CCGR0, CG2.

*kCLOCK\_Asrc* CCGR0, CG3.

*kCLOCK\_Caamsm* CCGR0, CG4 , caam\_secure\_mem clock.

*kCLOCK\_Caamwa* CCGR0, CG5 , caam\_wrapper\_aclk clock.  
*kCLOCK\_Caamwi* CCGR0, CG6 , caam\_wrapper\_ipg clock.  
*kCLOCK\_Can1* CCGR0, CG7.  
*kCLOCK\_Can1S* CCGR0, CG8 , Serial Clock.  
*kCLOCK\_Can2* CCGR0, CG9.  
*kCLOCK\_Can2S* CCGR0, CG10, Serial Clock.  
*kCLOCK\_CpuDbg* CCGR0, CG11.  
*kCLOCK\_Gpt2* CCGR0, CG12.  
*kCLOCK\_Gpt2S* CCGR0, CG13, Serial Clock.  
*kCLOCK\_Uart2* CCGR0, CG14.  
*kCLOCK\_Gpio2* CCGR0, CG15. CCM CCGR1  
*kCLOCK\_Ecspi1* CCGR1, CG0.  
*kCLOCK\_Ecspi2* CCGR1, CG1.  
*kCLOCK\_Ecspi3* CCGR1, CG2.  
*kCLOCK\_Ecspi4* CCGR1, CG3.  
*kCLOCK\_Adc\_5hc* CCGR1, CG4.  
*kCLOCK\_Uart3* CCGR1, CG5.  
*kCLOCK\_Epit1* CCGR1, CG6.  
*kCLOCK\_Epit2* CCGR1, CG7.  
*kCLOCK\_Adc1* CCGR1, CG8.  
*kCLOCK\_SimS* CCGR1, CG9.  
*kCLOCK\_Gpt1* CCGR1, CG10.  
*kCLOCK\_Gpt1S* CCGR1, CG11, Serial Clock.  
*kCLOCK\_Uart4* CCGR1, CG12.  
*kCLOCK\_Gpio1* CCGR1, CG13.  
*kCLOCK\_Csu* CCGR1, CG14.  
*kCLOCK\_Gpio5* CCGR1, CG15. CCM CCGR2 CCGR2, CG0 reserved  
*kCLOCK\_Csi* CCGR2, CG1. CCGR2, CG2 reserved  
*kCLOCK\_I2c1S* CCGR2, CG3, Serial Clock.  
*kCLOCK\_I2c2S* CCGR2, CG4, Serial Clock.  
*kCLOCK\_I2c3S* CCGR2, CG5, Serial Clock.  
*kCLOCK\_Ocotp* CCGR2, CG6.  
*kCLOCK\_IomuxcIpt* CCGR2, CG7.  
*kCLOCK\_Ipmux1* CCGR2, CG8.  
*kCLOCK\_Ipmux2* CCGR2, CG9.  
*kCLOCK\_Ipmux3* CCGR2, CG10.  
*kCLOCK\_Ipsync* CCGR2, CG11. CCGR2, CG12 reserved  
*kCLOCK\_Gpio3* CCGR2, CG13.  
*kCLOCK\_Lcd* CCGR2, CG14.  
*kCLOCK\_Pxp* CCGR2, CG15. CCM CCGR3  
*kCLOCK\_CsiMclk* CCGR3, CG0.  
*kCLOCK\_Uart5* CCGR3, CG1.  
*kCLOCK\_Enet* CCGR3, CG2.  
*kCLOCK\_Uart6* CCGR3, CG3.  
*kCLOCK\_Dap* CCGR3, CG4.

## Enumeration Type Documentation

*kCLOCK\_Lcdif1* CCGR3, CG5.  
*kCLOCK\_Gpio4* CCGR3, CG6.  
*kCLOCK\_Qspi1* CCGR3, CG7.  
*kCLOCK\_Wdog1* CCGR3, CG8.  
*kCLOCK\_Patch* CCGR3, CG9.  
*kCLOCK\_MmdcAClk* CCGR3, CG10. CCGR3, CG11 reserved  
*kCLOCK\_MmdcIpgP0* CCGR3, CG12.  
*kCLOCK\_MmdcIpgP1* CCGR3, CG13.  
*kCLOCK\_Axi* CCGR3, CG14. CCGR3, CG15 reserved CCM CCGR4 CCGR4, CG0 reserved  
*kCLOCK\_Iomuxc* CCGR4, CG1.  
*kCLOCK\_IomuxcGpr* CCGR4, CG2.  
*kCLOCK\_SimCpu* CCGR4, CG3.  
*kCLOCK\_ApbSlave* CCGR4, CG4.  
*kCLOCK\_Tsc* CCGR4, CG5.  
*kCLOCK\_SimM* CCGR4, CG6.  
*kCLOCK\_Axi2Apb* CCGR4, CG7.  
*kCLOCK\_Pwm1* CCGR4, CG8.  
*kCLOCK\_Pwm2* CCGR4, CG9.  
*kCLOCK\_Pwm3* CCGR4, CG10.  
*kCLOCK\_Pwm4* CCGR4, CG11.  
*kCLOCK\_RawNandBchApb* CCGR4, CG12.  
*kCLOCK\_RawNandBch* CCGR4, CG13.  
*kCLOCK\_RawNandGpmi* CCGR4, CG14.  
*kCLOCK\_RawNandGpmiApb* CCGR4, CG15. CCM CCGR5  
*kCLOCK\_Rom* CCGR5, CG0.  
*kCLOCK\_Stcr* CCGR5, CG1.  
*kCLOCK\_SnvsDryice* CCGR5, CG2.  
*kCLOCK\_Sdma* CCGR5, CG3.  
*kCLOCK\_Kpp* CCGR5, CG4.  
*kCLOCK\_Wdog2* CCGR5, CG5.  
*kCLOCK\_Spba* CCGR5, CG6.  
*kCLOCK\_Spdif* CCGR5, CG7.  
*kCLOCK\_SimMain* CCGR5, CG8.  
*kCLOCK\_SnvsHp* CCGR5, CG9.  
*kCLOCK\_SnvsLp* CCGR5, CG10.  
*kCLOCK\_Sai3* CCGR5, CG11.  
*kCLOCK\_Uart1* CCGR5, CG12.  
*kCLOCK\_Uart7* CCGR5, CG13.  
*kCLOCK\_Sai1* CCGR5, CG14.  
*kCLOCK\_Sai2* CCGR5, CG15. CCM CCGR6  
*kCLOCK\_UsbOh3* CCGR6, CG0.  
*kCLOCK\_Usdhc1* CCGR6, CG1.  
*kCLOCK\_Usdhc2* CCGR6, CG2.  
*kCLOCK\_Sim1* CCGR6, CG3.  
*kCLOCK\_Sim2* CCGR6, CG4.

*kCLOCK\_EimSlow* CCGR6, CG5. CCGR6, CG6 reserved  
*kCLOCK\_Uart8* CCGR6, CG7.  
*kCLOCK\_Pwm8* CCGR6, CG8. CCGR6, CG9 reserved  
*kCLOCK\_Wdog3* CCGR6, CG10.  
*kCLOCK\_Anadig* CCGR6, CG11.  
*kCLOCK\_I2c4S* CCGR6, CG12, Serial Clock.  
*kCLOCK\_Pwm5* CCGR6, CG13.  
*kCLOCK\_Pwm6* CCGR6, CG14.  
*kCLOCK\_Pwm7* CCGR6, CG15.

### 31.6.3 enum clock\_osc\_t

Enumerator

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

### 31.6.4 enum clock\_gate\_value\_t

Enumerator

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

### 31.6.5 enum clock\_mode\_t

Enumerator

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

### 31.6.6 enum clock\_mux\_t

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

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

## Enumeration Type Documentation

Enumerator

*kCLOCK\_StepMux* atep clock mux name  
*kCLOCK\_SecMux* secondary clock mux name  
*kCLOCK\_Pll1SwMux* pll1\_sw\_clk mux name  
*kCLOCK\_Pll3SwMux* Pll3\_sw\_clk mux name.  
*kCLOCK\_Periph2Mux* periph2 mux name  
*kCLOCK\_PeriphMux* periph mux name  
*kCLOCK\_AxiAltMux* axi alt mux name  
*kCLOCK\_AxiMux* axi mux name  
*kCLOCK\_PrePeriph2Mux* pre-periph2 mux name  
*kCLOCK\_PrePeriphMux* pre-periph mux name  
*kCLOCK\_Periph2Clk2Mux* periph2 clock2 mux name  
*kCLOCK\_PeriphClk2Mux* periph clock2 mux name  
*kCLOCK\_EimSlowMux* aclk eim slow mux name  
*kCLOCK\_GpmiMux* gpmi mux name  
*kCLOCK\_BchMux* bch mux name  
*kCLOCK\_Usdhc2Mux* usdhc2 mux name  
*kCLOCK\_Usdhc1Mux* usdhc1 mux name  
*kCLOCK\_Sai3Mux* sai3 mux name  
*kCLOCK\_Sai2Mux* sai2 mux name  
*kCLOCK\_Sai1Mux* sai1 mux name  
*kCLOCK\_Qspi1Mux* qspi1 mux name  
*kCLOCK\_PerclkMux* perclk mux name  
*kCLOCK\_VidMux* vid mux name  
*kCLOCK\_CanMux* can mux name  
*kCLOCK\_UartMux* uart mux name  
*kCLOCK\_EnfcMux* enfc mux name  
*kCLOCK\_LdbDi0Mux* ldb di0 mux name  
*kCLOCK\_SpdifMux* spdif mux name  
*kCLOCK\_SimPreMux* sim pre mux name  
*kCLOCK\_SimMux* sim mux name  
*kCLOCK\_EcspiMux* ecspi mux name  
*kCLOCK\_Lcdif1PreMux* lcdif1 pre mux name  
*kCLOCK\_Lcdif1Mux* lcdif1 mux name  
*kCLOCK\_CsiMux* csi mux name

### 31.6.7 enum clock\_div\_t

These constants define div control names for clock div setting.

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

Enumerator

- kCLOCK\_ArmDiv* core div name
- kCLOCK\_PeriphClk2Div* periph clock2 div name
- kCLOCK\_Periph2Clk2Div* periph2 clock2 div name
- kCLOCK\_AxiDiv* axi div name
- kCLOCK\_AhbDiv* ahb div name
- kCLOCK\_IpgDiv* ipg div name
- kCLOCK\_FabricMmdcDiv* mmdc/fabric div name
- kCLOCK\_Lcdif1Div* lcdif1 div name
- kCLOCK\_Qspi1Div* qspi1 div name
- kCLOCK\_EimSlowDiv* eim slow div name
- kCLOCK\_PerclkDiv* perclk div name
- kCLOCK\_VidDiv* vid div name
- kCLOCK\_VidPreDiv* vid pre div name
- kCLOCK\_LdbDi0Div* ldb di0 div name
- kCLOCK\_LdbDi1Div* ldb di1 div name
- kCLOCK\_CanDiv* can div name
- kCLOCK\_GpmiDiv* gpmi div name
- kCLOCK\_BchDiv* bch div name
- kCLOCK\_Usdhc2Div* usdhc2 div name
- kCLOCK\_Usdhc1Div* usdhc1 div name
- kCLOCK\_UartDiv* uart div name
- kCLOCK\_Sai3PreDiv* sai3 pre div name
- kCLOCK\_Sai3Div* sai3 div name
- kCLOCK\_Sai1PreDiv* sai1 pre div name
- kCLOCK\_Sai1Div* sai1 div name
- kCLOCK\_EnfcPreDiv* enfc pre div name
- kCLOCK\_EnfcDiv* enfc div name
- kCLOCK\_Sai2PreDiv* sai2 pre div name
- kCLOCK\_Sai2Div* sai2 div name
- kCLOCK\_Spdif0PreDiv* spdif pre div name
- kCLOCK\_Spdif0Div* spdif div name
- kCLOCK\_SimDiv* sim div name
- kCLOCK\_EcspiDiv* ecspi div name
- kCLOCK\_Lcdif1PreDiv* lcdif1 pre div name
- kCLOCK\_CsiDiv* csi div name

### 31.6.8 enum clock\_pll\_t

Enumerator

- kCLOCK\_PllArm* PLL ARM.
- kCLOCK\_PllSys* PLL SYS.

## Function Documentation

*kCLOCK\_PllUsb1* PLL USB1.  
*kCLOCK\_PllAudio* PLL Audio.  
*kCLOCK\_PllVideo* PLL Video.  
*kCLOCK\_PllEnet0* PLL Enet0.  
*kCLOCK\_PllEnet1* PLL Enet1.  
*kCLOCK\_PllEnet2* PLL Enet2.  
*kCLOCK\_PllUsb2* PLL USB2.

### 31.6.9 enum clock\_pfd\_t

Enumerator

*kCLOCK\_Pfd0* PLL PFD0.  
*kCLOCK\_Pfd1* PLL PFD1.  
*kCLOCK\_Pfd2* PLL PFD2.  
*kCLOCK\_Pfd3* PLL PFD3.

### 31.6.10 enum clock\_usb\_src\_t

Enumerator

*kCLOCK\_Usb480M* Use 480M.  
*kCLOCK\_UsbSrcUnused* Used when the function does not care the clock source.

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

Enumerator

*kCLOCK\_Usbphy480M* Use 480M.

## 31.7 Function Documentation

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

Parameters

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

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

Parameters

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

Returns

Clock mux value.

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

Parameters

<i>divider</i>	Which div node to set, see <a href="#">clock_div_t</a> .
<i>value</i>	Clock div value to set, different divider has different value range.

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

Parameters

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

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

## Function Documentation

Parameters

<i>name</i>	Which clock to enable, see <a href="#">clock_ip_name_t</a> .
<i>value</i>	Clock gate value to set, see <a href="#">clock_gate_value_t</a> .

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

Parameters

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

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

Parameters

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

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

Parameters

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

**31.7.9 uint32\_t CLOCK\_GetFreq ( [clock\\_name\\_t name](#) )**

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in [clock\\_name\\_t](#).

Parameters

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

Returns

Clock frequency value in hertz

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

This function supports two modes:

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

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

Parameters

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

Note

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

### 31.7.11 void CLOCK\_DeinitExternalClk ( void )

This function disables the external 24MHz clock.

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

### 31.7.12 void CLOCK\_SwitchOsc ( `clock_osc_t osc` )

This function switches the OSC source for SoC.

Parameters

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

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

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

## Function Documentation

Parameters

<i>osc</i>	OSC type to get frequency.
------------	----------------------------

Returns

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

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

Returns

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

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

Parameters

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

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

Parameters

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

### 31.7.17 void CLOCK\_InitArmPll( const clock\_arm\_pll\_config\_t \* *config* )

This function initialize the ARM PLL with specific settings

Parameters

<i>config</i>	configuration to set to PLL.
---------------	------------------------------

### 31.7.18 void CLOCK\_InitSysPll ( **const clock\_sys\_pll\_config\_t \* config** )

This function initializes the System PLL with specific settings

Parameters

<i>config</i>	Configuration to set to PLL.
---------------	------------------------------

### 31.7.19 void CLOCK\_InitUsb1Pll ( **const clock\_usb\_pll\_config\_t \* config** )

This function initializes the USB1 PLL with specific settings

Parameters

<i>config</i>	Configuration to set to PLL.
---------------	------------------------------

### 31.7.20 void CLOCK\_InitUsb2Pll ( **const clock\_usb\_pll\_config\_t \* config** )

This function initializes the USB2 PLL with specific settings

Parameters

<i>config</i>	Configuration to set to PLL.
---------------	------------------------------

### 31.7.21 void CLOCK\_InitAudioPll ( **const clock\_audio\_pll\_config\_t \* config** )

This function initializes the Audio PLL with specific settings

Parameters

<i>config</i>	Configuration to set to PLL.
---------------	------------------------------

### 31.7.22 void CLOCK\_InitVideoPll ( **const clock\_video\_pll\_config\_t \* config** )

This function configures the Video PLL with specific settings

## Function Documentation

Parameters

<i>config</i>	configuration to set to PLL.
---------------	------------------------------

### 31.7.23 void CLOCK\_InitEnetPll ( const clock\_enet\_pll\_config\_t \* *config* )

This function initializes the ENET PLL with specific settings.

Parameters

<i>config</i>	Configuration to set to PLL.
---------------	------------------------------

### 31.7.24 void CLOCK\_DeinitEnetPll ( void )

This function disables the ENET PLL.

### 31.7.25 uint32\_t CLOCK\_GetPllFreq ( clock\_pll\_t *pll* )

This function get current output frequency of specific PLL

Parameters

<i>pll</i>	pll name to get frequency.
------------	----------------------------

Returns

The PLL output frequency in hertz.

### 31.7.26 void CLOCK\_InitSysPfd ( clock\_pfd\_t *pfd*, uint8\_t *pfdFrac* )

This function initializes the System PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

Parameters

<i>pfd</i>	Which PFD clock to enable.
<i>pfdFrac</i>	The PFD FRAC value.

## Note

It is recommended that PFD settings are kept between 12-35.

**31.7.27 void CLOCK\_DeinitSysPfd ( clock\_pfd\_t *pfd* )**

This function disables the System PLL PFD.

## Parameters

<i>pfd</i>	Which PFD clock to disable.
------------	-----------------------------

**31.7.28 void CLOCK\_InitUsb1Pfd ( clock\_pfd\_t *pfd*, uint8\_t *pfdFrac* )**

This function initializes the USB1 PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

## Parameters

<i>pfd</i>	Which PFD clock to enable.
<i>pfdFrac</i>	The PFD FRAC value.

## Note

It is recommended that PFD settings are kept between 12-35.

**31.7.29 void CLOCK\_DeinitUsb1Pfd ( clock\_pfd\_t *pfd* )**

This function disables the USB1 PLL PFD.

## Parameters

<i>pfd</i>	Which PFD clock to disable.
------------	-----------------------------

**31.7.30 uint32\_t CLOCK\_GetSysPfdFreq ( clock\_pfd\_t *pfd* )**

This function get current output frequency of specific System PLL PFD

## Function Documentation

Parameters

<i>pfd</i>	pfd name to get frequency.
------------	----------------------------

Returns

The PFD output frequency in hertz.

### 31.7.31 **uint32\_t CLOCK\_GetUsb1PfdFreq ( clock\_pfd\_t *pfd* )**

This function get current output frequency of specific USB1 PLL PFD

Parameters

<i>pfd</i>	pfd name to get frequency.
------------	----------------------------

Returns

The PFD output frequency in hertz.

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

This function only enables the access to USB HS prepheral, upper layer should first call the [CLOCK\\_EnableUsbhs0PhyPllClock](#) to enable the PHY clock to use USB HS.

Parameters

<i>src</i>	USB HS does not care about the clock source, here must be <a href="#">kCLOCK_UsbSrcUnused</a> .
<i>freq</i>	USB HS does not care about the clock source, so this parameter is ignored.

Return values

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

### 31.7.33 **bool CLOCK\_EnableUsbhs0PhyPllClock ( clock\_usb\_phy\_src\_t *src*, uint32\_t *freq* )**

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

<i>src</i>	USB HS PHY PLL clock source.
<i>freq</i>	The frequency specified by <i>src</i> .

Return values

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

### 31.7.34 void CLOCK\_DisableUsbhs0PhyPllClock ( void )

This function disables USB HS PHY PLL clock.

### 31.7.35 bool CLOCK\_EnableUsbhs1Clock ( *clock\_usb\_src\_t src*, *uint32\_t freq* )

This function only enables the access to USB HS prepheral, upper layer should first call the [CLOCK\\_EnableUsbhs0PhyPllClock](#) to enable the PHY clock to use USB HS.

Parameters

<i>src</i>	USB HS does not care about the clock source, here must be <a href="#">kCLOCK_UsbSrcUnused</a> .
<i>freq</i>	USB HS does not care about the clock source, so this parameter is ignored.

Return values

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

### 31.7.36 bool CLOCK\_EnableUsbhs1PhyPllClock ( *clock\_usb\_phy\_src\_t src*, *uint32\_t freq* )

This function enables the internal 480MHz USB PHY PLL clock.

## Variable Documentation

Parameters

<i>src</i>	USB HS PHY PLL clock source.
<i>freq</i>	The frequency specified by src.

Return values

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

### 31.7.37 void CLOCK\_DisableUsbhs1PhyPllClock ( void )

This function disables USB HS PHY PLL clock.

## 31.8 Variable Documentation

### 31.8.1 uint32\_t g\_xtalFreq

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

```
* CLOCK_InitExternalClk(false); // Setup the 24M OSC/SYSOSC
* CLOCK_SetXtalFreq(240000000); // Set the XTAL value to clock driver.
*
```

### 31.8.2 uint32\_t g\_rtcXtalFreq

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

# Chapter 32

## Debug Console

### 32.1 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.

### 32.2 Function groups

#### 32.2.1 Initialization

To initialize the debug console, call the DbgConsole\_Init() function with these parameters. This function automatically enables the module and the clock.

```
/*
 * @brief Initializes the the peripheral used to debug messages.
 *
 * @param baseAddr      Indicates which address of the peripheral is used to send debug messages.
 * @param baudRate     The desired baud rate in bits per second.
 * @param device        Low level device type for the debug console, can be one of:
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_UART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPUART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPSCI,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_USBCDC.
 * @param clkSrcFreq   Frequency of peripheral source clock.
 *
 * @return             Whether initialization was successful or not.
 */
status_t DbgConsole_Init(uint32_t baseAddr, uint32_t baudRate, uint8_t device, uint32_t clkSrcFreq)
```

Selects the supported debug console hardware device type, such as

```
DEBUG_CONSOLE_DEVICE_TYPE_NONE
DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
DEBUG_CONSOLE_DEVICE_TYPE_UART
DEBUG_CONSOLE_DEVICE_TYPE_LPUART
DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the debug\_console\_state\_t structure, such as shown here.

```
typedef struct DebugConsoleState
{
    uint8_t                  type;
    void*                   base;
    debug_console_ops_t     ops;
} debug_console_state_t;
```

## Function groups

This example shows how to call the DbgConsole\_Init() given the user configuration structure.

```
uint32_t uartClkSrcFreq = CLOCK_GetFreq(BOARD_DEBUG_UART_CLKSRC);  
  
DbgConsole_Init(BOARD_DEBUG_UART_BASEADDR, BOARD_DEBUG_UART_BAUDRATE, DEBUG_CONSOLE_DEVICE_TYPE_UART,  
                 uartClkSrcFreq);
```

### 32.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
Do not support	

specifier	Description
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

## 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 */
```

### 32.3 Typical use case

#### Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

## Typical use case

### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\r\nTime: %u ticks %2.5f milliseconds\r\nDONE\r", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

### Print out failure messages using KSDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file,
           line, func);
    for (;;)
    {}
}
```

### Note:

To use 'printf' and 'scanf' for GNUC Base, add file '**fsl\_sbrk.c**' in path: ..\{package}\devices\{subset}\utilities\fsl-sbrk.c to your project.

## Modules

- Semihosting

## 32.4 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism 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.

### 32.4.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This 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. Start the project by choosing Project>Download and Debug.
4. Choose View>Terminal I/O to display the output from the I/O operations.

### 32.4.2 Guide Semihosting for Keil µVision

**NOTE:** Keil supports Semihosting only for Cortex-M3/Cortex-M4 cores.

#### Step 1: Prepare code

Remove function fputc and fgetc is used to support KEIL in "fsl\_debug\_console.c" and add the following code to project.

```
#pragma import(__use_no_semihosting_swi)

volatile int ITM_RxBuffer = ITM_RXBUFFER_EMPTY; /* used for Debug Input */
```

```
struct __FILE
{
    int handle;
};

FILE __stdout;
FILE __stdin;

int fputc(int ch, FILE *f)
{
    return (ITM_SendChar(ch));
}

int fgetc(FILE *f)
{ /* blocking */
    while (ITM_CheckChar() != 1)
        ;
    return (ITM_ReceiveChar());
}

int ferror(FILE *f)
{
    /* Your implementation of ferror */
    return EOF;
}

void _ttywrch(int ch)
{
    ITM_SendChar(ch);
}

void _sys_exit(int return_code)
{
label:
    goto label; /* endless loop */
}
```

### Step 2: Setting up the environment

1. In menu bar, choose Project>Options for target or using Alt+F7 or click.
2. Select "Target" tab and not select "Use MicroLIB".
3. Select "Debug" tab, select "J-Link/J-Trace Cortex" and click "Setting button".
4. Select "Debug" tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click OK.

### 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.

### 32.4.3 Guide Semihosting for KDS

**NOTE:** After the setting use "printf" for debugging.

#### Step 1: Setting up the environment

1. In menu bar, choose Project>Properties>C/C++ Build>Settings>Tool Settings.
2. Select “Libraries” on “Cross ARM C Linker” and delete “nosys”.
3. Select “Miscellaneous” on “Cross ARM C Linker”, add “-specs=rdimon.specs” to “Other link flags” and tick “Use newlib-nano”, and click OK.

#### Step 2: Building the project

1. In menu bar, choose Project>Build Project.

#### Step 3: Starting semihosting

1. In Debug configurations, choose "Startup" tab, tick “Enable semihosting and Telnet”. Press “Apply” and “Debug”.
2. After clicking Debug, the Window is displayed same as below. Run line by line to see the result in the Console Window.

### 32.4.4 Guide Semihosting for ATL

**NOTE:** J-Link has to be used to enable semihosting.

#### Step 1: Prepare code

Add the following code to the project.

```
int _write(int file, char *ptr, int len)
{
    /* Implement your write code here. This is used by puts and printf. */
    int i=0;
    for(i=0 ; i<len ; i++)
        ITM_SendChar((*ptr++));
    return len;
}
```

#### Step 2: Setting up the environment

1. In menu bar, choose Debug Configurations. In tab "Embedded C/C++ Application" choose "- Semihosting\_ATL\_xxx debug J-Link".
2. In tab "Debugger" set up as follows.
  - JTAG mode must be selected

## Semihosting

- SWV tracing must be enabled
  - Enter the Core Clock frequency, which is hardware board-specific.
  - Enter the desired SWO Clock frequency. The latter depends on the JTAG Probe and must be a multiple of the Core Clock value.
3. Click "Apply" and "Debug".

### Step 3: Starting semihosting

1. In the Views menu, expand the submenu SWV and open the docking view "SWV Console". 2. Open the SWV settings panel by clicking the "Configure Serial Wire Viewer" button in the SWV Console view toolbar. 3. Configure the data ports to be traced by enabling the ITM channel 0 check-box in the ITM stimulus ports group: Choose "EXETRC: Trace Exceptions" and In tab "ITM Stimulus Ports" choose "Enable Port" 0. Then click "OK".
2. It is recommended not to enable other SWV trace functionalities at the same time because this may over use the SWO pin causing packet loss due to a limited bandwidth (certain other SWV tracing capabilities can send a lot of data at very high-speed). Save the SWV configuration by clicking the OK button. The configuration is saved with other debug configurations and remains effective until changed.
3. Press the red Start/Stop Trace button to send the SWV configuration to the target board to enable SWV trace recording. The board does not send any SWV packages until it is properly configured. The SWV Configuration must be present, if the configuration registers on the target board are reset. Also, tracing does not start until the target starts to execute.
4. Start the target execution again by pressing the green Resume Debug button.
5. The SWV console now shows the printf() output.

### 32.4.5 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telnet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__stack_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__stack_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
```

```
defsym=__heap_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__heap_size__=0x2000")
```

## Step 2: Building the project

1. Change "CMakeLists.txt":

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"  
**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

### Replace paragraph

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fno-common")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-ffunction-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fdata-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-ffreestanding")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fno-builtin")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-mthumb")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-mapcs")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
--gc-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-static")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
muldefs")
```

### To

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
```

## Semihosting

```
G} --specs=rdimon.specs ")  
Remove  
target_link_libraries(semihosting_ARMGCC.elf debug nosys)  
2. Run "build_debug.bat" to build project
```

### Step 3: Starting semihosting

- (a) Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug  
d:  
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe  
target remote localhost:2331  
monitor reset  
monitor semihosting enable  
monitor semihosting thumbSWI 0xAB  
monitor semihosting IOClient 1  
monitor flash device = 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.

**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, ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2017 NXP B.V.

Document Number: MCUXSDKIMX6ULAPIRM  
Rev. 0  
Mar 2017

