

# MCUXpresso SDK API Reference Manual

**NXP Semiconductors**

Document Number: MCUXSDKIMX8MQAPIRM

Rev. 0

Dec 2018





# Contents

Chapter [Introduction](#)

Chapter [Driver errors status](#)

Chapter [Architectural Overview](#)

Chapter [Trademarks](#)

Chapter [Clock Driver](#)

<b>5.1</b>	<b><a href="#">Overview</a></b>	<b>11</b>
<b>5.2</b>	<b><a href="#">Data Structure Documentation</a></b>	<b>23</b>
5.2.1	<a href="#">struct osc_config_t</a>	23
5.2.2	<a href="#">struct ccm_analog_frac_pll_config_t</a>	24
5.2.3	<a href="#">struct ccm_analog_sscg_pll_config_t</a>	24
<b>5.3</b>	<b><a href="#">Macro Definition Documentation</a></b>	<b>25</b>
5.3.1	<a href="#">FSL_CLOCK_DRIVER_VERSION</a>	25
5.3.2	<a href="#">ECSPI_CLOCKS</a>	25
5.3.3	<a href="#">GPIO_CLOCKS</a>	25
5.3.4	<a href="#">GPT_CLOCKS</a>	25
5.3.5	<a href="#">I2C_CLOCKS</a>	25
5.3.6	<a href="#">IOMUX_CLOCKS</a>	26
5.3.7	<a href="#">PWM_CLOCKS</a>	26
5.3.8	<a href="#">RDC_CLOCKS</a>	26
5.3.9	<a href="#">SAI_CLOCKS</a>	26
5.3.10	<a href="#">RDC_SEMA42_CLOCKS</a>	26
5.3.11	<a href="#">UART_CLOCKS</a>	27
5.3.12	<a href="#">USDHC_CLOCKS</a>	27
5.3.13	<a href="#">WDOG_CLOCKS</a>	27
5.3.14	<a href="#">TMU_CLOCKS</a>	27
5.3.15	<a href="#">SDMA_CLOCKS</a>	27
5.3.16	<a href="#">MU_CLOCKS</a>	28
5.3.17	<a href="#">QSPI_CLOCKS</a>	28
5.3.18	<a href="#">kCLOCK_CoreSysClk</a>	28
5.3.19	<a href="#">CLOCK_GetCoreSysClkFreq</a>	28

# Contents

Section Number	Title	Page Number
<b>5.4</b>	<b>Enumeration Type Documentation</b>	<b>28</b>
5.4.1	clock_name_t	28
5.4.2	clock_ip_name_t	28
5.4.3	clock_root_control_t	30
5.4.4	clock_rootmux_m4_clk_sel_t	31
5.4.5	clock_rootmux_axi_clk_sel_t	31
5.4.6	clock_rootmux_ahb_clk_sel_t	31
5.4.7	clock_rootmux_qspi_clk_sel_t	32
5.4.8	clock_rootmux_ecspi_clk_sel_t	32
5.4.9	clock_rootmux_i2c_clk_sel_t	32
5.4.10	clock_rootmux_uart_clk_sel_t	33
5.4.11	clock_rootmux_gpt_t	33
5.4.12	clock_rootmux_wdog_clk_sel_t	33
5.4.13	clock_rootmux_Pwm_clk_sel_t	34
5.4.14	clock_rootmux_sai_clk_sel_t	34
5.4.15	clock_rootmux_noc_clk_sel_t	34
5.4.16	clock_pll_gate_t	35
5.4.17	clock_gate_value_t	35
5.4.18	clock_pll_bypass_ctrl_t	36
5.4.19	clock_pll_clke_t	36
5.4.20	_osc_mode	37
5.4.21	osc32_src_t	37
5.4.22	_ccm_analog_pll_ref_clk	37
<b>5.5</b>	<b>Function Documentation</b>	<b>38</b>
5.5.1	CLOCK_SetRootMux	38
5.5.2	CLOCK_GetRootMux	38
5.5.3	CLOCK_EnableRoot	38
5.5.4	CLOCK_DisableRoot	39
5.5.5	CLOCK_IsRootEnabled	39
5.5.6	CLOCK_UpdateRoot	39
5.5.7	CLOCK_SetRootDivider	40
5.5.8	CLOCK_GetRootPreDivider	40
5.5.9	CLOCK_GetRootPostDivider	40
5.5.10	CLOCK_InitOSC25M	41
5.5.11	CLOCK_DeinitOSC25M	41
5.5.12	CLOCK_InitOSC27M	41
5.5.13	CLOCK_DeinitOSC27M	41
5.5.14	CLOCK_SwitchOSC32Src	41
5.5.15	CLOCK_ControlGate	41
5.5.16	CLOCK_EnableClock	42
5.5.17	CLOCK_DisableClock	42
5.5.18	CLOCK_PowerUpPll	42
5.5.19	CLOCK_PowerDownPll	42
5.5.20	CLOCK_SetPllBypass	43

# Contents

Section Number	Title	Page Number
5.5.21	<a href="#">CLOCK_IsPllBypassed</a>	43
5.5.22	<a href="#">CLOCK_IsPllLocked</a>	43
5.5.23	<a href="#">CLOCK_EnableAnalogClock</a>	44
5.5.24	<a href="#">CLOCK_DisableAnalogClock</a>	44
5.5.25	<a href="#">CLOCK_OverrideAnalogClke</a>	44
5.5.26	<a href="#">CLOCK_OverridePllPd</a>	45
5.5.27	<a href="#">CLOCK_InitArmPll</a>	45
5.5.28	<a href="#">CLOCK_InitSysPll1</a>	45
5.5.29	<a href="#">CLOCK_InitSysPll2</a>	46
5.5.30	<a href="#">CLOCK_InitSysPll3</a>	46
5.5.31	<a href="#">CLOCK_InitDramPll</a>	46
5.5.32	<a href="#">CLOCK_InitAudioPll1</a>	47
5.5.33	<a href="#">CLOCK_InitAudioPll2</a>	47
5.5.34	<a href="#">CLOCK_InitVideoPll1</a>	47
5.5.35	<a href="#">CLOCK_InitVideoPll2</a>	48
5.5.36	<a href="#">CLOCK_InitSSCGPll</a>	48
5.5.37	<a href="#">CLOCK_GetSSCGPllFreq</a>	48
5.5.38	<a href="#">CLOCK_InitFracPll</a>	49
5.5.39	<a href="#">CLOCK_GetFracPllFreq</a>	49
5.5.40	<a href="#">CLOCK_GetPllFreq</a>	49
5.5.41	<a href="#">CLOCK_GetPllRefClkFreq</a>	50
5.5.42	<a href="#">CLOCK_GetFreq</a>	50
5.5.43	<a href="#">CLOCK_GetCoreM4Freq</a>	50
5.5.44	<a href="#">CLOCK_GetAxiFreq</a>	51
5.5.45	<a href="#">CLOCK_GetAhbFreq</a>	51

## Chapter ECSPI: Serial Peripheral Interface Driver

<b>6.1</b>	<b>Overview</b>	<b>53</b>
<b>6.2</b>	<b>ECSPI Driver</b>	<b>54</b>
6.2.1	Overview	54
6.2.2	Typical use case	54
6.2.3	Data Structure Documentation	59
6.2.4	Macro Definition Documentation	61
6.2.5	Enumeration Type Documentation	61
6.2.6	Function Documentation	64
<b>6.3</b>	<b>ECSPI FreeRTOS Driver</b>	<b>76</b>
6.3.1	Overview	76
6.3.2	Macro Definition Documentation	76
6.3.3	Function Documentation	76

# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>GPT: General Purpose Timer</b>	
<b>7.1</b>	<b>Overview</b>	<b>79</b>
<b>7.2</b>	<b>Function groups</b>	<b>79</b>
7.2.1	Initialization and deinitialization	79
<b>7.3</b>	<b>Typical use case</b>	<b>79</b>
7.3.1	GPT interrupt example	79
<b>7.4</b>	<b>Data Structure Documentation</b>	<b>82</b>
7.4.1	struct gpt_config_t	82
<b>7.5</b>	<b>Enumeration Type Documentation</b>	<b>83</b>
7.5.1	gpt_clock_source_t	83
7.5.2	gpt_input_capture_channel_t	83
7.5.3	gpt_input_operation_mode_t	84
7.5.4	gpt_output_compare_channel_t	84
7.5.5	gpt_output_operation_mode_t	84
7.5.6	gpt_interrupt_enable_t	84
7.5.7	gpt_status_flag_t	85
<b>7.6</b>	<b>Function Documentation</b>	<b>85</b>
7.6.1	GPT_Init	85
7.6.2	GPT_Deinit	85
7.6.3	GPT_GetDefaultConfig	85
7.6.4	GPT_SoftwareReset	86
7.6.5	GPT_SetClockSource	86
7.6.6	GPT_GetClockSource	86
7.6.7	GPT_SetClockDivider	86
7.6.8	GPT_GetClockDivider	87
7.6.9	GPT_SetOscClockDivider	87
7.6.10	GPT_GetOscClockDivider	87
7.6.11	GPT_StartTimer	87
7.6.12	GPT_StopTimer	88
7.6.13	GPT_GetCurrentTimerCount	88
7.6.14	GPT_SetInputOperationMode	88
7.6.15	GPT_GetInputOperationMode	88
7.6.16	GPT_GetInputCaptureValue	89
7.6.17	GPT_SetOutputOperationMode	89
7.6.18	GPT_GetOutputOperationMode	90
7.6.19	GPT_SetOutputCompareValue	90
7.6.20	GPT_GetOutputCompareValue	90
7.6.21	GPT_ForceOutput	91
7.6.22	GPT_EnableInterrupts	91
7.6.23	GPT_DisableInterrupts	91

# Contents

Section Number	Title	Page Number
7.6.24	GPT_GetEnabledInterrupts . . . . .	91
7.6.25	GPT_GetStatusFlags . . . . .	92
7.6.26	GPT_ClearStatusFlags . . . . .	92
<b>Chapter</b>	<b>GPIO: General-Purpose Input/Output Driver</b>	
<b>8.1</b>	<b>Overview . . . . .</b>	<b>93</b>
<b>8.2</b>	<b>GPIO Driver . . . . .</b>	<b>94</b>
8.2.1	Overview . . . . .	94
8.2.2	Typical use case . . . . .	94
8.2.3	Data Structure Documentation . . . . .	96
8.2.4	Macro Definition Documentation . . . . .	96
8.2.5	Enumeration Type Documentation . . . . .	96
8.2.6	Function Documentation . . . . .	97
<b>Chapter</b>	<b>I2C: Inter-Integrated Circuit Driver</b>	
<b>9.1</b>	<b>Overview . . . . .</b>	<b>103</b>
<b>9.2</b>	<b>I2C Driver . . . . .</b>	<b>104</b>
9.2.1	Overview . . . . .	104
9.2.2	Typical use case . . . . .	104
9.2.3	Data Structure Documentation . . . . .	108
9.2.4	Macro Definition Documentation . . . . .	112
9.2.5	Typedef Documentation . . . . .	112
9.2.6	Enumeration Type Documentation . . . . .	112
9.2.7	Function Documentation . . . . .	114
<b>9.3</b>	<b>I2C FreeRTOS Driver . . . . .</b>	<b>127</b>
9.3.1	Overview . . . . .	127
9.3.2	Macro Definition Documentation . . . . .	127
9.3.3	Function Documentation . . . . .	127
<b>Chapter</b>	<b>PWM: Pulse Width Modulation Driver</b>	
<b>10.1</b>	<b>Overview . . . . .</b>	<b>131</b>
<b>10.2</b>	<b>PWM Driver . . . . .</b>	<b>132</b>
10.2.1	Overview . . . . .	132
10.2.2	Typical use case . . . . .	132
10.2.3	Enumeration Type Documentation . . . . .	135
10.2.4	Function Documentation . . . . .	137

# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>UART: Universal Asynchronous Receiver/Transmitter Driver</b>	
<b>11.1</b>	<b>Overview</b>	<b>143</b>
<b>11.2</b>	<b>UART Driver</b>	<b>144</b>
11.2.1	Overview	144
11.2.2	Typical use case	144
11.2.3	Data Structure Documentation	151
11.2.4	Macro Definition Documentation	153
11.2.5	Typedef Documentation	153
11.2.6	Enumeration Type Documentation	153
11.2.7	Function Documentation	155
11.2.8	Variable Documentation	171
<b>11.3</b>	<b>UART FreeRTOS Driver</b>	<b>172</b>
11.3.1	Overview	172
11.3.2	Data Structure Documentation	172
11.3.3	Macro Definition Documentation	173
11.3.4	Function Documentation	173
<b>Chapter</b>	<b>MU: Messaging Unit</b>	
<b>12.1</b>	<b>Overview</b>	<b>175</b>
<b>12.2</b>	<b>Function description</b>	<b>175</b>
12.2.1	MU initialization	175
12.2.2	MU message	175
12.2.3	MU flags	176
12.2.4	Status and interrupt	176
12.2.5	MU misc functions	176
<b>12.3</b>	<b>Macro Definition Documentation</b>	<b>179</b>
12.3.1	FSL_MU_DRIVER_VERSION	179
<b>12.4</b>	<b>Enumeration Type Documentation</b>	<b>179</b>
12.4.1	_mu_status_flags	179
12.4.2	_mu_interrupt_enable	179
12.4.3	_mu_interrupt_trigger	180
<b>12.5</b>	<b>Function Documentation</b>	<b>180</b>
12.5.1	MU_Init	180
12.5.2	MU_Deinit	180
12.5.3	MU_SendMsgNonBlocking	180
12.5.4	MU_SendMsg	181
12.5.5	MU_ReceiveMsgNonBlocking	181
12.5.6	MU_ReceiveMsg	182



# Contents

Section Number	Title	Page Number
12.5.7	MU_SetFlagsNonBlocking . . . . .	182
12.5.8	MU_SetFlags . . . . .	183
12.5.9	MU_GetFlags . . . . .	183
12.5.10	MU_GetStatusFlags . . . . .	183
12.5.11	MU_ClearStatusFlags . . . . .	184
12.5.12	MU_EnableInterrupts . . . . .	185
12.5.13	MU_DisableInterrupts . . . . .	185
12.5.14	MU_TriggerInterrupts . . . . .	185
12.5.15	MU_ClearNmi . . . . .	186
12.5.16	MU_BootCoreB . . . . .	186
12.5.17	MU_HoldCoreBReset . . . . .	186
12.5.18	MU_BootOtherCore . . . . .	187
12.5.19	MU_HoldOtherCoreReset . . . . .	187
12.5.20	MU_ResetBothSides . . . . .	187
12.5.21	MU_HardwareResetOtherCore . . . . .	188
12.5.22	MU_SetClockOnOtherCoreEnable . . . . .	189
12.5.23	MU_GetOtherCorePowerMode . . . . .	189

## Chapter **QSPI: Quad Serial Peripheral Interface Driver**

<b>13.1</b>	<b>Overview . . . . .</b>	<b>191</b>
<b>13.2</b>	<b>Data Structure Documentation . . . . .</b>	<b>195</b>
13.2.1	struct qspi_dqs_config_t . . . . .	195
13.2.2	struct qspi_flash_timing_t . . . . .	196
13.2.3	struct qspi_config_t . . . . .	196
13.2.4	struct qspi_flash_config_t . . . . .	197
13.2.5	struct qspi_transfer_t . . . . .	198
<b>13.3</b>	<b>Macro Definition Documentation . . . . .</b>	<b>198</b>
13.3.1	FSL_QSPI_DRIVER_VERSION . . . . .	198
<b>13.4</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>198</b>
13.4.1	_status_t . . . . .	198
13.4.2	qspi_read_area_t . . . . .	198
13.4.3	qspi_command_seq_t . . . . .	198
13.4.4	qspi_fifo_t . . . . .	199
13.4.5	qspi_endianness_t . . . . .	199
13.4.6	_qspi_error_flags . . . . .	199
13.4.7	_qspi_flags . . . . .	199
13.4.8	_qspi_interrupt_enable . . . . .	200
13.4.9	_qspi_dma_enable . . . . .	201
13.4.10	qspi_dqs_phrase_shift_t . . . . .	201
<b>13.5</b>	<b>Function Documentation . . . . .</b>	<b>201</b>

# Contents

Section Number	Title	Page Number
13.5.1	QSPI_GetInstance . . . . .	201
13.5.2	QSPI_Init . . . . .	201
13.5.3	QSPI_GetDefaultQspiConfig . . . . .	201
13.5.4	QSPI_Deinit . . . . .	202
13.5.5	QSPI_SetFlashConfig . . . . .	202
13.5.6	QSPI_SoftwareReset . . . . .	202
13.5.7	QSPI_Enable . . . . .	202
13.5.8	QSPI_GetStatusFlags . . . . .	203
13.5.9	QSPI_GetErrorStatusFlags . . . . .	203
13.5.10	QSPI_ClearErrorFlag . . . . .	203
13.5.11	QSPI_EnableInterrupts . . . . .	204
13.5.12	QSPI_DisableInterrupts . . . . .	204
13.5.13	QSPI_EnableDMA . . . . .	204
13.5.14	QSPI_GetTxDataRegisterAddress . . . . .	204
13.5.15	QSPI_GetRxDataRegisterAddress . . . . .	205
13.5.16	QSPI_SetIPCommandAddress . . . . .	205
13.5.17	QSPI_SetIPCommandSize . . . . .	205
13.5.18	QSPI_ExecuteIPCommand . . . . .	206
13.5.19	QSPI_ExecuteAHBCommand . . . . .	206
13.5.20	QSPI_EnableIPParallelMode . . . . .	206
13.5.21	QSPI_EnableAHBParallelMode . . . . .	206
13.5.22	QSPI_UpdateLUT . . . . .	207
13.5.23	QSPI_ClearFifo . . . . .	207
13.5.24	QSPI_ClearCommandSequence . . . . .	207
13.5.25	QSPI_EnableDDRMode . . . . .	207
13.5.26	QSPI_SetReadDataArea . . . . .	208
13.5.27	QSPI_WriteBlocking . . . . .	208
13.5.28	QSPI_WriteData . . . . .	208
13.5.29	QSPI_ReadBlocking . . . . .	209
13.5.30	QSPI_ReadData . . . . .	209
13.5.31	QSPI_TransferSendBlocking . . . . .	209
13.5.32	QSPI_TransferReceiveBlocking . . . . .	210

## Chapter RDC: Resource Domain Controller

14.1	Overview . . . . .	211
14.2	Data Structure Documentation . . . . .	213
14.2.1	struct rdc_hardware_config_t . . . . .	213
14.2.2	struct rdc_domain_assignment_t . . . . .	213
14.2.3	struct rdc_periph_access_config_t . . . . .	213
14.2.4	struct rdc_mem_access_config_t . . . . .	214
14.2.5	struct rdc_mem_status_t . . . . .	215
14.3	Macro Definition Documentation . . . . .	215

# Contents

Section Number	Title	Page Number
14.3.1	FSL_RDC_DRIVER_VERSION . . . . .	215
<b>14.4</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>215</b>
14.4.1	_rdc_interrupts . . . . .	215
14.4.2	_rdc_flags . . . . .	216
14.4.3	_rdc_access_policy . . . . .	216
<b>14.5</b>	<b>Function Documentation . . . . .</b>	<b>216</b>
14.5.1	RDC_Init . . . . .	216
14.5.2	RDC_Deinit . . . . .	216
14.5.3	RDC_GetHardwareConfig . . . . .	216
14.5.4	RDC_EnableInterrupts . . . . .	217
14.5.5	RDC_DisableInterrupts . . . . .	217
14.5.6	RDC_GetInterruptStatus . . . . .	217
14.5.7	RDC_ClearInterruptStatus . . . . .	217
14.5.8	RDC_GetStatus . . . . .	218
14.5.9	RDC_ClearStatus . . . . .	218
14.5.10	RDC_SetMasterDomainAssignment . . . . .	218
14.5.11	RDC_GetDefaultMasterDomainAssignment . . . . .	219
14.5.12	RDC_LockMasterDomainAssignment . . . . .	219
14.5.13	RDC_SetPeriphAccessConfig . . . . .	219
14.5.14	RDC_GetDefaultPeriphAccessConfig . . . . .	219
14.5.15	RDC_LockPeriphAccessConfig . . . . .	220
14.5.16	RDC_SetMemAccessConfig . . . . .	220
14.5.17	RDC_GetDefaultMemAccessConfig . . . . .	220
14.5.18	RDC_LockMemAccessConfig . . . . .	221
14.5.19	RDC_SetMemAccessValid . . . . .	221
14.5.20	RDC_GetMemViolationStatus . . . . .	221
14.5.21	RDC_ClearMemViolationFlag . . . . .	222
14.5.22	RDC_GetCurrentMasterDomainId . . . . .	222
<b>Chapter</b>	<b>RDC_SEMA42: Hardware Semaphores Driver</b>	
<b>15.1</b>	<b>Overview . . . . .</b>	<b>223</b>
<b>15.2</b>	<b>Macro Definition Documentation . . . . .</b>	<b>224</b>
15.2.1	RDC_SEMA42_GATE_NUM_RESET_ALL . . . . .	224
15.2.2	RDC_SEMA42_GATEn . . . . .	224
15.2.3	RDC_SEMA42_GATE_COUNT . . . . .	224
<b>15.3</b>	<b>Function Documentation . . . . .</b>	<b>224</b>
15.3.1	RDC_SEMA42_Init . . . . .	224
15.3.2	RDC_SEMA42_Deinit . . . . .	224
15.3.3	RDC_SEMA42_TryLock . . . . .	225
15.3.4	RDC_SEMA42_Lock . . . . .	225

# Contents

Section Number	Title	Page Number
15.3.5	RDC_SEMA42_Unlock . . . . .	226
15.3.6	RDC_SEMA42_GetLockMasterIndex . . . . .	226
15.3.7	RDC_SEMA42_GetLockDomainID . . . . .	226
15.3.8	RDC_SEMA42_ResetGate . . . . .	227
15.3.9	RDC_SEMA42_ResetAllGates . . . . .	228
<b>Chapter</b>	<b>SAI: Serial Audio Interface</b>	
<b>16.1</b>	<b>Overview . . . . .</b>	<b>229</b>
<b>16.2</b>	<b>Typical use case . . . . .</b>	<b>229</b>
16.2.1	SAI Send/receive using an interrupt method . . . . .	229
16.2.2	SAI Send/receive using a DMA method . . . . .	229
<b>16.3</b>	<b>Data Structure Documentation . . . . .</b>	<b>236</b>
16.3.1	struct sai_config_t . . . . .	236
16.3.2	struct sai_transfer_format_t . . . . .	236
16.3.3	struct sai_transfer_t . . . . .	237
16.3.4	struct _sai_handle . . . . .	237
<b>16.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>238</b>
16.4.1	SAI_XFER_QUEUE_SIZE . . . . .	238
<b>16.5</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>238</b>
16.5.1	_sai_status_t . . . . .	238
16.5.2	_sai_channel_mask . . . . .	238
16.5.3	sai_protocol_t . . . . .	238
16.5.4	sai_master_slave_t . . . . .	239
16.5.5	sai_mono_stereo_t . . . . .	239
16.5.6	sai_data_order_t . . . . .	239
16.5.7	sai_clock_polarity_t . . . . .	239
16.5.8	sai_sync_mode_t . . . . .	239
16.5.9	sai_mclk_source_t . . . . .	240
16.5.10	sai_bclk_source_t . . . . .	240
16.5.11	_sai_interrupt_enable_t . . . . .	240
16.5.12	_sai_dma_enable_t . . . . .	240
16.5.13	_sai_flags . . . . .	241
16.5.14	sai_reset_type_t . . . . .	241
16.5.15	sai_fifo_packing_t . . . . .	241
16.5.16	sai_sample_rate_t . . . . .	241
16.5.17	sai_word_width_t . . . . .	242
<b>16.6</b>	<b>Function Documentation . . . . .</b>	<b>242</b>
16.6.1	SAI_TxInit . . . . .	242
16.6.2	SAI_RxInit . . . . .	242

# Contents

Section Number	Title	Page Number
16.6.3	SAI_TxGetDefaultConfig . . . . .	243
16.6.4	SAI_RxGetDefaultConfig . . . . .	243
16.6.5	SAI_Deinit . . . . .	243
16.6.6	SAI_TxReset . . . . .	244
16.6.7	SAI_RxReset . . . . .	244
16.6.8	SAI_TxEnable . . . . .	244
16.6.9	SAI_RxEnable . . . . .	244
16.6.10	SAI_TxGetStatusFlag . . . . .	244
16.6.11	SAI_TxClearStatusFlags . . . . .	245
16.6.12	SAI_RxGetStatusFlag . . . . .	245
16.6.13	SAI_RxClearStatusFlags . . . . .	245
16.6.14	SAI_TxSoftwareReset . . . . .	246
16.6.15	SAI_RxSoftwareReset . . . . .	246
16.6.16	SAI_TxSetChannelFIFOMask . . . . .	246
16.6.17	SAI_RxSetChannelFIFOMask . . . . .	247
16.6.18	SAI_TxSetDataOrder . . . . .	247
16.6.19	SAI_RxSetDataOrder . . . . .	247
16.6.20	SAI_TxSetBitClockPolarity . . . . .	247
16.6.21	SAI_RxSetBitClockPolarity . . . . .	248
16.6.22	SAI_TxSetFrameSyncPolarity . . . . .	248
16.6.23	SAI_RxSetFrameSyncPolarity . . . . .	248
16.6.24	SAI_TxSetFIFOPacking . . . . .	248
16.6.25	SAI_RxSetFIFOPacking . . . . .	249
16.6.26	SAI_TxSetFIFOErrorContinue . . . . .	249
16.6.27	SAI_RxSetFIFOErrorContinue . . . . .	249
16.6.28	SAI_TxEnableInterrupts . . . . .	249
16.6.29	SAI_RxEnableInterrupts . . . . .	250
16.6.30	SAI_TxDisableInterrupts . . . . .	251
16.6.31	SAI_RxDisableInterrupts . . . . .	251
16.6.32	SAI_TxEnableDMA . . . . .	252
16.6.33	SAI_RxEnableDMA . . . . .	252
16.6.34	SAI_TxGetDataRegisterAddress . . . . .	252
16.6.35	SAI_RxGetDataRegisterAddress . . . . .	253
16.6.36	SAI_TxSetFormat . . . . .	253
16.6.37	SAI_RxSetFormat . . . . .	254
16.6.38	SAI_WriteBlocking . . . . .	254
16.6.39	SAI_WriteMultiChannelBlocking . . . . .	254
16.6.40	SAI_WriteData . . . . .	255
16.6.41	SAI_ReadBlocking . . . . .	255
16.6.42	SAI_ReadMultiChannelBlocking . . . . .	256
16.6.43	SAI_ReadData . . . . .	256
16.6.44	SAI_TransferTxCreateHandle . . . . .	256
16.6.45	SAI_TransferRxCreateHandle . . . . .	257
16.6.46	SAI_TransferTxSetFormat . . . . .	257
16.6.47	SAI_TransferRxSetFormat . . . . .	258

# Contents

Section Number	Title	Page Number
16.6.48	SAI_TransferSendNonBlocking . . . . .	258
16.6.49	SAI_TransferReceiveNonBlocking . . . . .	259
16.6.50	SAI_TransferGetSendCount . . . . .	259
16.6.51	SAI_TransferGetReceiveCount . . . . .	260
16.6.52	SAI_TransferAbortSend . . . . .	260
16.6.53	SAI_TransferAbortReceive . . . . .	261
16.6.54	SAI_TransferTerminateSend . . . . .	261
16.6.55	SAI_TransferTerminateReceive . . . . .	261
16.6.56	SAI_TransferTxHandleIRQ . . . . .	262
16.6.57	SAI_TransferRxHandleIRQ . . . . .	262
<b>16.7</b>	<b>SAI DMA Driver . . . . .</b>	<b>263</b>
<b>16.8</b>	<b>SAI EDMA Driver . . . . .</b>	<b>264</b>
<b>16.9</b>	<b>SAI SDMA Driver . . . . .</b>	<b>265</b>
<b>Chapter</b>	<b>SEMA4: Hardware Semaphores Driver</b>	
<b>17.1</b>	<b>Overview . . . . .</b>	<b>267</b>
<b>17.2</b>	<b>Macro Definition Documentation . . . . .</b>	<b>268</b>
17.2.1	SEMA4_GATE_NUM_RESET_ALL . . . . .	268
<b>17.3</b>	<b>Function Documentation . . . . .</b>	<b>268</b>
17.3.1	SEMA4_Init . . . . .	268
17.3.2	SEMA4_Deinit . . . . .	268
17.3.3	SEMA4_TryLock . . . . .	268
17.3.4	SEMA4_Lock . . . . .	269
17.3.5	SEMA4_Unlock . . . . .	269
17.3.6	SEMA4_GetLockProc . . . . .	269
17.3.7	SEMA4_ResetGate . . . . .	270
17.3.8	SEMA4_ResetAllGates . . . . .	270
17.3.9	SEMA4_EnableGateNotifyInterrupt . . . . .	271
17.3.10	SEMA4_DisableGateNotifyInterrupt . . . . .	271
17.3.11	SEMA4_GetGateNotifyStatus . . . . .	271
17.3.12	SEMA4_ResetGateNotify . . . . .	272
17.3.13	SEMA4_ResetAllGateNotify . . . . .	272
<b>Chapter</b>	<b>TMU: Thermal Management Unit Driver</b>	
<b>18.1</b>	<b>Overview . . . . .</b>	<b>275</b>
<b>18.2</b>	<b>Typical use case . . . . .</b>	<b>275</b>
18.2.1	Monitor and report Configuration . . . . .	275

# Contents

Section Number	Title	Page Number
<b>18.3</b>	<b>Data Structure Documentation</b>	<b>277</b>
18.3.1	struct tmu_thresold_config_t	277
18.3.2	struct tmu_interrupt_status_t	277
18.3.3	struct tmu_config_t	278
<b>18.4</b>	<b>Macro Definition Documentation</b>	<b>278</b>
18.4.1	FSL_TMU_DRIVER_VERSION	278
<b>18.5</b>	<b>Enumeration Type Documentation</b>	<b>278</b>
18.5.1	_tmu_interrupt_enable	278
18.5.2	_tmu_interrupt_status_flags	278
18.5.3	_tmu_status_flags	279
18.5.4	tmu_average_low_pass_filter_t	279
<b>18.6</b>	<b>Function Documentation</b>	<b>279</b>
18.6.1	TMU_Init	279
18.6.2	TMU_Deinit	279
18.6.3	TMU_GetDefaultConfig	280
18.6.4	TMU_Enable	280
18.6.5	TMU_EnableInterrupts	280
18.6.6	TMU_DisableInterrupts	280
18.6.7	TMU_GetInterruptStatusFlags	281
18.6.8	TMU_ClearInterruptStatusFlags	281
18.6.9	TMU_GetStatusFlags	281
18.6.10	TMU_GetHighestTemperature	281
18.6.11	TMU_GetLowestTemperature	282
18.6.12	TMU_GetImmediateTemperature	282
18.6.13	TMU_GetAverageTemperature	283
18.6.14	TMU_SetHighTemperatureThresold	284
<b>18.7</b>	<b>Variable Documentation</b>	<b>284</b>
18.7.1	immediateThresoldEnable	284
18.7.2	AverageThresoldEnable	284
18.7.3	AverageCriticalThresoldEnable	284
18.7.4	immediateThresoldValue	284
18.7.5	averageThresoldValue	284
18.7.6	averageCriticalThresoldValue	284
18.7.7	interruptDetectMask	284
18.7.8	immediateInterruptsSiteMask	285
18.7.9	AverageInterruptsSiteMask	285
18.7.10	AverageCriticalInterruptsSiteMask	285
18.7.11	monitorInterval	285
18.7.12	monitorSiteSelection	285
18.7.13	averageLPF	285

# Contents

Section Number	Title	Page Number
<b>Chapter</b>	<b>WDOG: Watchdog Timer Driver</b>	
<b>19.1</b>	<b>Overview</b>	<b>287</b>
<b>19.2</b>	<b>Typical use case</b>	<b>287</b>
<b>19.3</b>	<b>Data Structure Documentation</b>	<b>288</b>
19.3.1	struct wdog_work_mode_t	288
19.3.2	struct wdog_config_t	288
<b>19.4</b>	<b>Enumeration Type Documentation</b>	<b>289</b>
19.4.1	_wdog_interrupt_enable	289
19.4.2	_wdog_status_flags	289
<b>19.5</b>	<b>Function Documentation</b>	<b>289</b>
19.5.1	WDOG_GetDefaultConfig	289
19.5.2	WDOG_Init	290
19.5.3	WDOG_Deinit	290
19.5.4	WDOG_Enable	290
19.5.5	WDOG_Disable	291
19.5.6	WDOG_TriggerSystemSoftwareReset	291
19.5.7	WDOG_TriggerSoftwareSignal	291
19.5.8	WDOG_EnableInterrupts	292
19.5.9	WDOG_GetStatusFlags	293
19.5.10	WDOG_ClearInterruptStatus	293
19.5.11	WDOG_SetTimeoutValue	294
19.5.12	WDOG_SetInterruptTimeoutValue	294
19.5.13	WDOG_DisablePowerDownEnable	294
19.5.14	WDOG_Refresh	295
<b>Chapter</b>	<b>IOMUXC: Input/Output Multiplexing Controller</b>	
<b>20.1</b>	<b>Overview</b>	<b>297</b>
<b>20.2</b>	<b>Macro Definition Documentation</b>	<b>314</b>
20.2.1	FSL_IOMUXC_DRIVER_VERSION	314
<b>20.3</b>	<b>Function Documentation</b>	<b>314</b>
20.3.1	IOMUXC_SetPinMux	314
20.3.2	IOMUXC_SetPinConfig	315
<b>Chapter</b>	<b>Debug Console</b>	
<b>21.1</b>	<b>Overview</b>	<b>317</b>
<b>21.2</b>	<b>Function groups</b>	<b>317</b>



# Contents

Section Number	Title	Page Number
21.2.1	Initialization . . . . .	317
21.2.2	Advanced Feature . . . . .	317
<b>21.3</b>	<b>Typical use case . . . . .</b>	<b>321</b>
<b>21.4</b>	<b>Macro Definition Documentation . . . . .</b>	<b>323</b>
21.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN . . . . .	323
21.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK . . . . .	324
21.4.3	DEBUGCONSOLE_DISABLE . . . . .	324
21.4.4	SDK_DEBUGCONSOLE . . . . .	324
21.4.5	SDK_DEBUGCONSOLE_UART . . . . .	324
21.4.6	PRINTF . . . . .	324
<b>21.5</b>	<b>Function Documentation . . . . .</b>	<b>324</b>
21.5.1	DbgConsole_Init . . . . .	324
21.5.2	DbgConsole_Deinit . . . . .	325
21.5.3	DbgConsole_Printf . . . . .	325
21.5.4	DbgConsole_Putchar . . . . .	325
21.5.5	DbgConsole_Scanf . . . . .	326
21.5.6	DbgConsole_Getchar . . . . .	326
21.5.7	DbgConsole_Flush . . . . .	326
21.5.8	StrFormatPrintf . . . . .	327
21.5.9	StrFormatScanf . . . . .	327
<b>21.6</b>	<b>Semihosting . . . . .</b>	<b>328</b>
21.6.1	Guide Semihosting for IAR . . . . .	328
21.6.2	Guide Semihosting for Keil $\mu$ Vision . . . . .	328
21.6.3	Guide Semihosting for MCUXpresso IDE . . . . .	329
21.6.4	Guide Semihosting for ARMGCC . . . . .	329
<b>21.7</b>	<b>SWO . . . . .</b>	<b>332</b>
21.7.1	Guide SWO for SDK . . . . .	332
21.7.2	Guide SWO for Keil $\mu$ Vision . . . . .	333
21.7.3	Guide SWO for MCUXpresso IDE . . . . .	334
21.7.4	Guide SWO for ARMGCC . . . . .	334



# Chapter 1

## Introduction

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

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

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

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

- IAR Embedded Workbench
- GNU Arm Embedded Toolchain

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

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

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

Table 2: MCUXpresso SDK Folder Structure

## Chapter 2

### Driver errors status

- [kStatus\\_ECSPI\\_Busy](#) = 6400
- [kStatus\\_ECSPI\\_Idle](#) = 6401
- [kStatus\\_ECSPI\\_Error](#) = 6402
- [kStatus\\_ECSPI\\_HardwareOverflow](#) = 6403
- [kStatus\\_I2C\\_Busy](#) = 1100
- [kStatus\\_I2C\\_Idle](#) = 1101
- [kStatus\\_I2C\\_Nak](#) = 1102
- [kStatus\\_I2C\\_ArbitrationLost](#) = 1103
- [kStatus\\_I2C\\_Timeout](#) = 1104
- [kStatus\\_I2C\\_Addr\\_Nak](#) = 1105
- [kStatus\\_UART\\_TxBusy](#) = 2800
- [kStatus\\_UART\\_RxBusy](#) = 2801
- [kStatus\\_UART\\_TxIdle](#) = 2802
- [kStatus\\_UART\\_RxIdle](#) = 2803
- [kStatus\\_UART\\_TxWatermarkTooLarge](#) = 2804
- [kStatus\\_UART\\_RxWatermarkTooLarge](#) = 2805
- [kStatus\\_UART\\_FlagCannotClearManually](#) = 2806
- [kStatus\\_UART\\_Error](#) = 2807
- [kStatus\\_UART\\_RxRingBufferOverrun](#) = 2808
- [kStatus\\_UART\\_RxHardwareOverrun](#) = 2809
- [kStatus\\_UART\\_NoiseError](#) = 2810
- [kStatus\\_UART\\_FramingError](#) = 2811
- [kStatus\\_UART\\_ParityError](#) = 2812
- [kStatus\\_UART\\_BaudrateNotSupport](#) = 2813
- [kStatus\\_UART\\_BreakDetect](#) = 2814
- [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



## 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

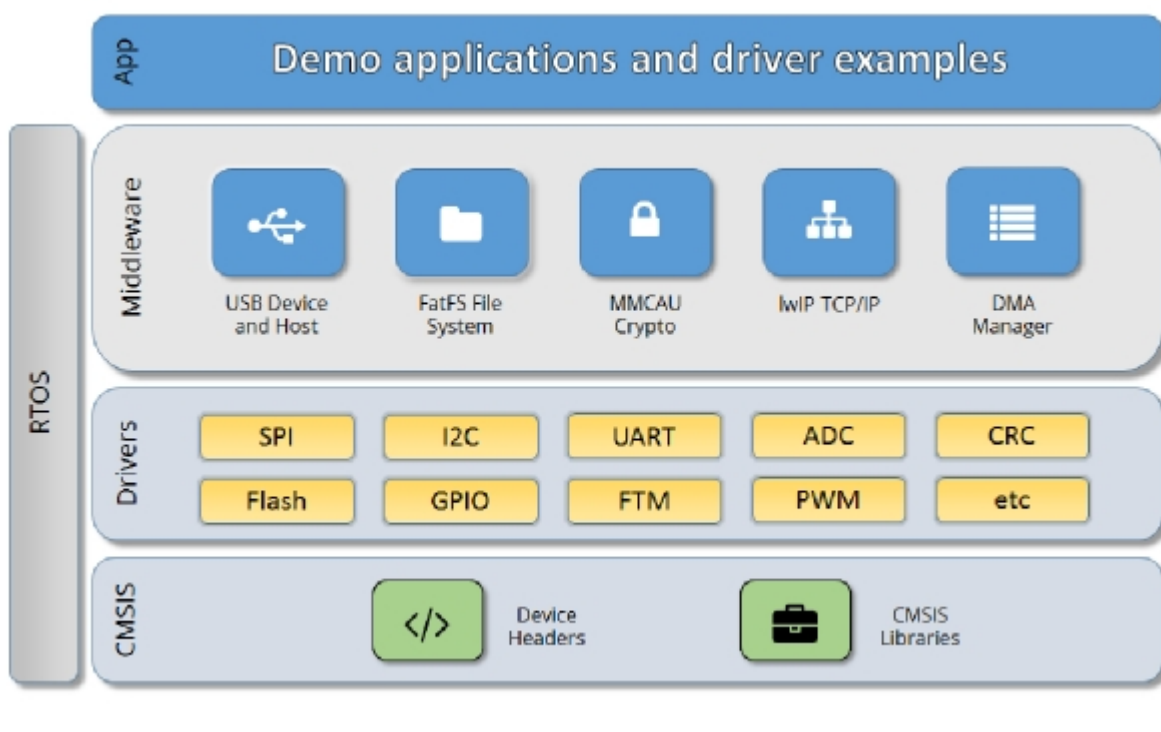


Figure 1: MCUXpresso SDK Block Diagram

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

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

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

## Feature Header Files

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

## Application

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



## Chapter 4

### Trademarks

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

How to Reach Us:

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

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

NXP 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: {<http://www.nxp.com/SalesTermsandConditions>} {[nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions)}.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.



## Chapter 5 Clock Driver

### 5.1 Overview

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

### Data Structures

- struct [osc\\_config\\_t](#)  
*OSC configuration structure. [More...](#)*
- struct [ccm\\_analog\\_frac\\_pll\\_config\\_t](#)  
*Fractional-N PLL configuration. [More...](#)*
- struct [ccm\\_analog\\_sscg\\_pll\\_config\\_t](#)  
*SSCG PLL configuration. [More...](#)*

### Macros

- #define [OSC25M\\_CLK\\_FREQ](#) 25000000U  
*XTAL 25M clock frequency.*
- #define [OSC27M\\_CLK\\_FREQ](#) 27000000U  
*XTAL 27M clock frequency.*
- #define [HDMI\\_PHY\\_27M\\_FREQ](#) 27000000U  
*HDMI PHY 27M clock frequency.*
- #define [CLKPN\\_FREQ](#) 0U  
*clock1PN frequency.*
- #define [ECSPI\\_CLOCKS](#)  
*Clock ip name array for ECSPI.*
- #define [GPIO\\_CLOCKS](#)  
*Clock ip name array for GPIO.*
- #define [GPT\\_CLOCKS](#)  
*Clock ip name array for GPT.*
- #define [I2C\\_CLOCKS](#)  
*Clock ip name array for I2C.*
- #define [IOMUX\\_CLOCKS](#)  
*Clock ip name array for IOMUX.*
- #define [PWM\\_CLOCKS](#)  
*Clock ip name array for PWM.*
- #define [RDC\\_CLOCKS](#)  
*Clock ip name array for RDC.*
- #define [SAI\\_CLOCKS](#)  
*Clock ip name array for SAI.*
- #define [RDC\\_SEMA42\\_CLOCKS](#)  
*Clock ip name array for RDC SEMA42.*
- #define [UART\\_CLOCKS](#)  
*Clock ip name array for UART.*
- #define [USDHC\\_CLOCKS](#)

## Overview

- *Clock ip name array for USDHC.*  
• #define [WDOG\\_CLOCKS](#)
- *Clock ip name array for WDOG.*  
• #define [TMU\\_CLOCKS](#)
- *Clock ip name array for TEMPSENSOR.*  
• #define [SDMA\\_CLOCKS](#)
- *Clock ip name array for SDMA.*  
• #define [MU\\_CLOCKS](#)
- *Clock ip name array for MU.*  
• #define [QSPI\\_CLOCKS](#)
- *Clock ip name array for QSPI.*  
• #define [CCM\\_BIT\\_FIELD\\_EXTRACTION](#)(val, mask, shift) (((val)&mask) >> shift)
- *CCM reg macros to extract corresponding registers bit field.*  
• #define [CCM\\_REG\\_OFF](#)(root, off) (\*((volatile uint32\_t \*)((uint32\_t)root + off)))
- *CCM reg macros to map corresponding registers.*  
• #define [CCM\\_ANALOG\\_TUPLE](#)(reg, shift) (((uint32\_t)((CCM\_ANALOG\_Type \*)0U)->reg & 0xFFFFU) << 16U) | (shift))
- *CCM ANALOG tuple macros to map corresponding registers and bit fields.*  
• #define [CCM\\_TUPLE](#)(ccgr, root) (ccgr << 16U | root)
- *CCM CCGR and root tuple.*  
• #define [kCLOCK\\_CoreSysClk](#) [kCLOCK\\_CoreM4Clk](#)
- *For compatible with other platforms without CCM.*  
• #define [CLOCK\\_GetCoreSysClkFreq](#) [CLOCK\\_GetCoreM4Freq](#)
- *For compatible with other platforms without CCM.*

## Enumerations

- enum [clock\\_name\\_t](#) {  
    [kCLOCK\\_CoreM4Clk](#),  
    [kCLOCK\\_AxiClk](#),  
    [kCLOCK\\_AhbClk](#),  
    [kCLOCK\\_IpgClk](#) }  
    *Clock name used to get clock frequency.*
- enum [clock\\_ip\\_name\\_t](#) { ,

```

kCLOCK_Debug = CCM_TUPLE(4U, 32U),
kCLOCK_Dram = CCM_TUPLE(5U, 64U),
kCLOCK_Ecspi1 = CCM_TUPLE(7U, 101U),
kCLOCK_Ecspi2 = CCM_TUPLE(8U, 102U),
kCLOCK_Ecspi3 = CCM_TUPLE(9U, 131U),
kCLOCK_Gpio1 = CCM_TUPLE(11U, 33U),
kCLOCK_Gpio2 = CCM_TUPLE(12U, 33U),
kCLOCK_Gpio3 = CCM_TUPLE(13U, 33U),
kCLOCK_Gpio4 = CCM_TUPLE(14U, 33U),
kCLOCK_Gpio5 = CCM_TUPLE(15U, 33U),
kCLOCK_Gpt1 = CCM_TUPLE(16U, 107U),
kCLOCK_Gpt2 = CCM_TUPLE(17U, 108U),
kCLOCK_Gpt3 = CCM_TUPLE(18U, 109U),
kCLOCK_Gpt4 = CCM_TUPLE(19U, 110U),
kCLOCK_Gpt5 = CCM_TUPLE(20U, 111U),
kCLOCK_Gpt6 = CCM_TUPLE(21U, 112U),
kCLOCK_I2c1 = CCM_TUPLE(23U, 90U),
kCLOCK_I2c2 = CCM_TUPLE(24U, 91U),
kCLOCK_I2c3 = CCM_TUPLE(25U, 92U),
kCLOCK_I2c4 = CCM_TUPLE(26U, 93U),
kCLOCK_Iomux0 = CCM_TUPLE(27U, 33U),
kCLOCK_Iomux1 = CCM_TUPLE(28U, 33U),
kCLOCK_Iomux2 = CCM_TUPLE(29U, 33U),
kCLOCK_Iomux3 = CCM_TUPLE(30U, 33U),
kCLOCK_Iomux4 = CCM_TUPLE(31U, 33U),
kCLOCK_M4 = CCM_TUPLE(32U, 1U),
kCLOCK_Mu = CCM_TUPLE(33U, 33U),
kCLOCK_Ocram = CCM_TUPLE(35U, 16U),
kCLOCK_OcramS = CCM_TUPLE(36U, 32U),
kCLOCK_Pwm1 = CCM_TUPLE(40U, 103U),
kCLOCK_Pwm2 = CCM_TUPLE(41U, 104U),
kCLOCK_Pwm3 = CCM_TUPLE(42U, 105U),
kCLOCK_Pwm4 = CCM_TUPLE(43U, 106U),
kCLOCK_Qspi = CCM_TUPLE(47U, 87U),
kCLOCK_Rdc = CCM_TUPLE(49U, 33U),
kCLOCK_Sai1 = CCM_TUPLE(51U, 75U),
kCLOCK_Sai2 = CCM_TUPLE(52U, 76U),
kCLOCK_Sai3 = CCM_TUPLE(53U, 77U),
kCLOCK_Sai4 = CCM_TUPLE(54U, 78U),
kCLOCK_Sai5 = CCM_TUPLE(55U, 79U),
kCLOCK_Sai6 = CCM_TUPLE(56U, 80U),
kCLOCK_Sdma1 = CCM_TUPLE(58U, 33U),
kCLOCK_Sdma2 = CCM_TUPLE(59U, 35U),
kCLOCK_Sec_Debug = CCM_TUPLE(60U, 33U),
kCLOCK_Sema42_1 = CCM_TUPLE(61U, 33U),
kCLOCK_Sema42_2 = CCM_TUPLE(62U, 33U),
kCLOCK_Sim_display = CCM_TUPLE(63U, 33U),
kCLOCK_Sim_m = CCM_TUPLE(65U, 32U),
kCLOCK_Sim_main = CCM_TUPLE(66U, 16U),

```

`kCLOCK_TempSensor` = CCM\_TUPLE(98U, 0xFFFF) }

*CCM CCGR gate control.*

- enum `clock_root_control_t` {
  - `kCLOCK_RootM4` = (uint32\_t)(&(CCM)->ROOT[1].TARGET\_ROOT),
  - `kCLOCK_RootAxi` = (uint32\_t)(&(CCM)->ROOT[16].TARGET\_ROOT),
  - `kCLOCK_RootNoc` = (uint32\_t)(&(CCM)->ROOT[26].TARGET\_ROOT),
  - `kCLOCK_RootAhb` = (uint32\_t)(&(CCM)->ROOT[32].TARGET\_ROOT),
  - `kCLOCK_RootIpg` = (uint32\_t)(&(CCM)->ROOT[33].TARGET\_ROOT),
  - `kCLOCK_RootDramAlt` = (uint32\_t)(&(CCM)->ROOT[64].TARGET\_ROOT),
  - `kCLOCK_RootSai1` = (uint32\_t)(&(CCM)->ROOT[75].TARGET\_ROOT),
  - `kCLOCK_RootSai2` = (uint32\_t)(&(CCM)->ROOT[76].TARGET\_ROOT),
  - `kCLOCK_RootSai3` = (uint32\_t)(&(CCM)->ROOT[77].TARGET\_ROOT),
  - `kCLOCK_RootSai4` = (uint32\_t)(&(CCM)->ROOT[78].TARGET\_ROOT),
  - `kCLOCK_RootSai5` = (uint32\_t)(&(CCM)->ROOT[79].TARGET\_ROOT),
  - `kCLOCK_RootSai6` = (uint32\_t)(&(CCM)->ROOT[80].TARGET\_ROOT),
  - `kCLOCK_RootQspi` = (uint32\_t)(&(CCM)->ROOT[87].TARGET\_ROOT),
  - `kCLOCK_RootI2c1` = (uint32\_t)(&(CCM)->ROOT[90].TARGET\_ROOT),
  - `kCLOCK_RootI2c2` = (uint32\_t)(&(CCM)->ROOT[91].TARGET\_ROOT),
  - `kCLOCK_RootI2c3` = (uint32\_t)(&(CCM)->ROOT[92].TARGET\_ROOT),
  - `kCLOCK_RootI2c4` = (uint32\_t)(&(CCM)->ROOT[93].TARGET\_ROOT),
  - `kCLOCK_RootUart1` = (uint32\_t)(&(CCM)->ROOT[94].TARGET\_ROOT),
  - `kCLOCK_RootUart2` = (uint32\_t)(&(CCM)->ROOT[95].TARGET\_ROOT),
  - `kCLOCK_RootUart3` = (uint32\_t)(&(CCM)->ROOT[96].TARGET\_ROOT),
  - `kCLOCK_RootUart4` = (uint32\_t)(&(CCM)->ROOT[97].TARGET\_ROOT),
  - `kCLOCK_RootEcspl1` = (uint32\_t)(&(CCM)->ROOT[101].TARGET\_ROOT),
  - `kCLOCK_RootEcspl2` = (uint32\_t)(&(CCM)->ROOT[102].TARGET\_ROOT),
  - `kCLOCK_RootEcspl3` = (uint32\_t)(&(CCM)->ROOT[131].TARGET\_ROOT),
  - `kCLOCK_RootPwm1` = (uint32\_t)(&(CCM)->ROOT[103].TARGET\_ROOT),
  - `kCLOCK_RootPwm2` = (uint32\_t)(&(CCM)->ROOT[104].TARGET\_ROOT),
  - `kCLOCK_RootPwm3` = (uint32\_t)(&(CCM)->ROOT[105].TARGET\_ROOT),
  - `kCLOCK_RootPwm4` = (uint32\_t)(&(CCM)->ROOT[106].TARGET\_ROOT),
  - `kCLOCK_RootGpt1` = (uint32\_t)(&(CCM)->ROOT[107].TARGET\_ROOT),
  - `kCLOCK_RootGpt2` = (uint32\_t)(&(CCM)->ROOT[108].TARGET\_ROOT),
  - `kCLOCK_RootGpt3` = (uint32\_t)(&(CCM)->ROOT[109].TARGET\_ROOT),
  - `kCLOCK_RootGpt4` = (uint32\_t)(&(CCM)->ROOT[110].TARGET\_ROOT),
  - `kCLOCK_RootGpt5` = (uint32\_t)(&(CCM)->ROOT[111].TARGET\_ROOT),
  - `kCLOCK_RootGpt6` = (uint32\_t)(&(CCM)->ROOT[112].TARGET\_ROOT),
  - `kCLOCK_RootWdog` = (uint32\_t)(&(CCM)->ROOT[114].TARGET\_ROOT) }

*ccm root name used to get clock frequency.*

- enum `clock_rootmux_m4_clk_sel_t` {



```

kCLOCK_M4RootmuxOsc25m = 0U,
kCLOCK_M4RootmuxSysPll2Div5 = 1U,
kCLOCK_M4RootmuxSysPll2Div4 = 2U,
kCLOCK_M4RootmuxSysPll1Div3 = 3U,
kCLOCK_M4RootmuxSysPll1 = 4U,
kCLOCK_M4RootmuxAudioPll1 = 5U,
kCLOCK_M4RootmuxVideoPll1 = 6U,
kCLOCK_M4RootmuxSysPll3 = 7U }

```

*Root clock select enumeration for ARM Cortex-M4 core.*

- enum clock\_rootmux\_axi\_clk\_sel\_t {  
kCLOCK\_AxiRootmuxOsc25m = 0U,  
kCLOCK\_AxiRootmuxSysPll2Div3 = 1U,  
kCLOCK\_AxiRootmuxSysPll1 = 2U,  
kCLOCK\_AxiRootmuxSysPll2Div4 = 3U,  
kCLOCK\_AxiRootmuxSysPll2 = 4U,  
kCLOCK\_AxiRootmuxAudioPll1 = 5U,  
kCLOCK\_AxiRootmuxVideoPll1 = 6U,  
kCLOCK\_AxiRootmuxSysPll1Div8 = 7U }

*Root clock select enumeration for AXI bus.*

- enum clock\_rootmux\_ahb\_clk\_sel\_t {  
kCLOCK\_AhbRootmuxOsc25m = 0U,  
kCLOCK\_AhbRootmuxSysPll1Div6 = 1U,  
kCLOCK\_AhbRootmuxSysPll1 = 2U,  
kCLOCK\_AhbRootmuxSysPll1Div2 = 3U,  
kCLOCK\_AhbRootmuxSysPll2Div8 = 4U,  
kCLOCK\_AhbRootmuxSysPll3 = 5U,  
kCLOCK\_AhbRootmuxAudioPll1 = 6U,  
kCLOCK\_AhbRootmuxVideoPll1 = 7U }

*Root clock select enumeration for AHB bus.*

- enum clock\_rootmux\_qspi\_clk\_sel\_t {  
kCLOCK\_QspiRootmuxOsc25m = 0U,  
kCLOCK\_QspiRootmuxSysPll1Div2 = 1U,  
kCLOCK\_QspiRootmuxSysPll1 = 2U,  
kCLOCK\_QspiRootmuxSysPll2Div2 = 3U,  
kCLOCK\_QspiRootmuxAudioPll2 = 4,  
kCLOCK\_QspiRootmuxSysPll1Div3 = 5U,  
kCLOCK\_QspiRootmuxSysPll3 = 6U,  
kCLOCK\_QspiRootmuxSysPll1Div8 = 7U }

*Root clock select enumeration for QSPI peripheral.*

- enum clock\_rootmux\_ecspi\_clk\_sel\_t {

## Overview

```
kCLOCK_EcspiRootmuxOsc25m = 0U,  
kCLOCK_EcspiRootmuxSysPll2Div5 = 1U,  
kCLOCK_EcspiRootmuxSysPll1Div20 = 2U,  
kCLOCK_EcspiRootmuxSysPll1Div5 = 3U,  
kCLOCK_EcspiRootmuxSysPll1 = 4U,  
kCLOCK_EcspiRootmuxSysPll3 = 5U,  
kCLOCK_EcspiRootmuxSysPll2Div4 = 6U,  
kCLOCK_EcspiRootmuxAudioPll2 = 7U }
```

*Root clock select enumeration for ECSPi peripheral.*

- enum `clock_rootmux_i2c_clk_sel_t` {  
    kCLOCK\_I2cRootmuxOsc25m = 0U,  
    kCLOCK\_I2cRootmuxSysPll1Div5 = 1U,  
    kCLOCK\_I2cRootmuxSysPll2Div20 = 2U,  
    kCLOCK\_I2cRootmuxSysPll3 = 3U,  
    kCLOCK\_I2cRootmuxAudioPll1 = 4U,  
    kCLOCK\_I2cRootmuxVideoPll1 = 5U,  
    kCLOCK\_I2cRootmuxAudioPll2 = 6U,  
    kCLOCK\_I2cRootmuxSysPll1Div6 = 7U }

*Root clock select enumeration for I2C peripheral.*

- enum `clock_rootmux_uart_clk_sel_t` {  
    kCLOCK\_UartRootmuxOsc25m = 0U,  
    kCLOCK\_UartRootmuxSysPll1Div10 = 1U,  
    kCLOCK\_UartRootmuxSysPll2Div5 = 2U,  
    kCLOCK\_UartRootmuxSysPll2Div10 = 3U,  
    kCLOCK\_UartRootmuxSysPll3 = 4U,  
    kCLOCK\_UartRootmuxExtClk2 = 5U,  
    kCLOCK\_UartRootmuxExtClk34 = 6U,  
    kCLOCK\_UartRootmuxAudioPll2 = 7U }

*Root clock select enumeration for UART peripheral.*

- enum `clock_rootmux_gpt_t` {  
    kCLOCK\_GptRootmuxOsc25m = 0U,  
    kCLOCK\_GptRootmuxSystemPll2Div10 = 1U,  
    kCLOCK\_GptRootmuxSysPll1Div2 = 2U,  
    kCLOCK\_GptRootmuxSysPll1Div20 = 3U,  
    kCLOCK\_GptRootmuxVideoPll1 = 4U,  
    kCLOCK\_GptRootmuxSystemPll1Div10 = 5U,  
    kCLOCK\_GptRootmuxAudioPll1 = 6U,  
    kCLOCK\_GptRootmuxExtClk123 = 7U }

*Root clock select enumeration for GPT peripheral.*

- enum `clock_rootmux_wdog_clk_sel_t` {

```

kCLOCK_WdogRootmuxOsc25m = 0U,
kCLOCK_WdogRootmuxSysPll1Div6 = 1U,
kCLOCK_WdogRootmuxSysPll1Div5 = 2U,
kCLOCK_WdogRootmuxVpuPll = 3U,
kCLOCK_WdogRootmuxSystemPll2Div8 = 4U,
kCLOCK_WdogRootmuxSystemPll3 = 5U,
kCLOCK_WdogRootmuxSystemPll1Div10 = 6U,
kCLOCK_WdogRootmuxSystemPll2Div6 = 7U }

```

*Root clock select enumeration for WDOG peripheral.*

- enum `clock_rootmux_Pwm_clk_sel_t` {  
`kCLOCK_PwmRootmuxOsc25m = 0U,`  
`kCLOCK_PwmRootmuxSysPll2Div10 = 1U,`  
`kCLOCK_PwmRootmuxSysPll1Div5 = 2U,`  
`kCLOCK_PwmRootmuxSysPll1Div20 = 3U,`  
`kCLOCK_PwmRootmuxSystemPll3 = 4U,`  
`kCLOCK_PwmRootmuxExtClk12 = 5U,`  
`kCLOCK_PwmRootmuxSystemPll1Div10 = 6U,`  
`kCLOCK_PwmRootmuxVideoPll1 = 7U }`

*Root clock select enumeration for PWM peripheral.*

- enum `clock_rootmux_sai_clk_sel_t` {  
`kCLOCK_SaiRootmuxOsc25m = 0U,`  
`kCLOCK_SaiRootmuxAudioPll1 = 1U,`  
`kCLOCK_SaiRootmuxAudioPll2 = 2U,`  
`kCLOCK_SaiRootmuxVideoPll1 = 3U,`  
`kCLOCK_SaiRootmuxSysPll1Div6 = 4U,`  
`kCLOCK_SaiRootmuxOsc27m = 5U,`  
`kCLOCK_SaiRootmuxExtClk123 = 6U,`  
`kCLOCK_SaiRootmuxExtClk234 = 7U }`

*Root clock select enumeration for SAI peripheral.*

- enum `clock_rootmux_noc_clk_sel_t` {  
`kCLOCK_NocRootmuxOsc25m = 0U,`  
`kCLOCK_NocRootmuxSysPll1 = 1U,`  
`kCLOCK_NocRootmuxSysPll3 = 2U,`  
`kCLOCK_NocRootmuxSysPll2 = 3U,`  
`kCLOCK_NocRootmuxSysPll2Div2 = 4U,`  
`kCLOCK_NocRootmuxAudioPll1 = 5U,`  
`kCLOCK_NocRootmuxVideoPll1 = 6U,`  
`kCLOCK_NocRootmuxAudioPll2 = 7U }`

*Root clock select enumeration for NOC CLK.*

- enum `clock_pll_gate_t` {

```

kCLOCK_ArmPllGate = (uint32_t)(amp(ccm)->pll_ctrl[12].pll_ctrl),
kCLOCK_GpuPllGate = (uint32_t)(amp(ccm)->pll_ctrl[13].pll_ctrl),
kCLOCK_VpuPllGate = (uint32_t)(amp(ccm)->pll_ctrl[14].pll_ctrl),
kCLOCK_DramPllGate = (uint32_t)(amp(ccm)->pll_ctrl[15].pll_ctrl),
kCLOCK_SysPll1Gate = (uint32_t)(amp(ccm)->pll_ctrl[16].pll_ctrl),
kCLOCK_SysPll1Div2Gate = (uint32_t)(amp(ccm)->pll_ctrl[17].pll_ctrl),
kCLOCK_SysPll1Div3Gate = (uint32_t)(amp(ccm)->pll_ctrl[18].pll_ctrl),
kCLOCK_SysPll1Div4Gate = (uint32_t)(amp(ccm)->pll_ctrl[19].pll_ctrl),
kCLOCK_SysPll1Div5Gate = (uint32_t)(amp(ccm)->pll_ctrl[20].pll_ctrl),
kCLOCK_SysPll1Div6Gate = (uint32_t)(amp(ccm)->pll_ctrl[21].pll_ctrl),
kCLOCK_SysPll1Div8Gate = (uint32_t)(amp(ccm)->pll_ctrl[22].pll_ctrl),
kCLOCK_SysPll1Div10Gate = (uint32_t)(amp(ccm)->pll_ctrl[23].pll_ctrl),
kCLOCK_SysPll1Div20Gate = (uint32_t)(amp(ccm)->pll_ctrl[24].pll_ctrl),
kCLOCK_SysPll2Gate = (uint32_t)(amp(ccm)->pll_ctrl[25].pll_ctrl),
kCLOCK_SysPll2Div2Gate = (uint32_t)(amp(ccm)->pll_ctrl[26].pll_ctrl),
kCLOCK_SysPll2Div3Gate = (uint32_t)(amp(ccm)->pll_ctrl[27].pll_ctrl),
kCLOCK_SysPll2Div4Gate = (uint32_t)(amp(ccm)->pll_ctrl[28].pll_ctrl),
kCLOCK_SysPll2Div5Gate = (uint32_t)(amp(ccm)->pll_ctrl[29].pll_ctrl),
kCLOCK_SysPll2Div6Gate = (uint32_t)(amp(ccm)->pll_ctrl[30].pll_ctrl),
kCLOCK_SysPll2Div8Gate = (uint32_t)(amp(ccm)->pll_ctrl[31].pll_ctrl),
kCLOCK_SysPll2Div10Gate = (uint32_t)(amp(ccm)->pll_ctrl[32].pll_ctrl),
kCLOCK_SysPll2Div20Gate = (uint32_t)(amp(ccm)->pll_ctrl[33].pll_ctrl),
kCLOCK_SysPll3Gate = (uint32_t)(amp(ccm)->pll_ctrl[34].pll_ctrl),
kCLOCK_AudioPll1Gate = (uint32_t)(amp(ccm)->pll_ctrl[35].pll_ctrl),
kCLOCK_AudioPll2Gate = (uint32_t)(amp(ccm)->pll_ctrl[36].pll_ctrl),
kCLOCK_VideoPll1Gate = (uint32_t)(amp(ccm)->pll_ctrl[37].pll_ctrl),
kCLOCK_VideoPll2Gate = (uint32_t)(amp(ccm)->pll_ctrl[38].pll_ctrl) }

```

*CCM PLL gate control.*

- enum `clock_gate_value_t` {  
`kCLOCK_ClockNotNeeded` = 0x0U,  
`kCLOCK_ClockNeededRun` = 0x1111U,  
`kCLOCK_ClockNeededRunWait` = 0x2222U,  
`kCLOCK_ClockNeededAll` = 0x3333U }

*CCM gate control value.*

- enum `clock_pll_bypass_ctrl_t` {

```

kCLOCK_AudioPll1BypassCtrl,
kCLOCK_AudioPll2BypassCtrl,
kCLOCK_VideoPll1BypassCtrl,
kCLOCK_GpuPLLWrBypassCtrl,
kCLOCK_VpuPllPwrBypassCtrl,
kCLOCK_ArmPllPwrBypassCtrl,
kCLOCK_SysPll1InternalPll1BypassCtrl,
kCLOCK_SysPll1InternalPll2BypassCtrl,
kCLOCK_SysPll2InternalPll1BypassCtrl,
kCLOCK_SysPll2InternalPll2BypassCtrl,
kCLOCK_SysPll3InternalPll1BypassCtrl,
kCLOCK_SysPll3InternalPll2BypassCtrl,
kCLOCK_VideoPll2InternalPll1BypassCtrl,
kCLOCK_VideoPll2InternalPll2BypassCtrl,
kCLOCK_DramPllInternalPll1BypassCtrl,
kCLOCK_DramPllInternalPll2BypassCtrl }

```

*PLL control names for PLL bypass.*

- enum `clock_pll_clke_t` {
 

```

kCLOCK_AudioPll1Clke,
kCLOCK_AudioPll2Clke,
kCLOCK_VideoPll1Clke,
kCLOCK_GpuPllClke = CCM_ANALOG_TUPLE(GPU_PLL_CFG0, CCM_ANALOG_GPU_P-
LL_CFG0_PLL_CLKE_SHIFT),
kCLOCK_VpuPllClke = CCM_ANALOG_TUPLE(VPU_PLL_CFG0, CCM_ANALOG_VPU_P-
LL_CFG0_PLL_CLKE_SHIFT),
kCLOCK_ArmPllClke = CCM_ANALOG_TUPLE(ARM_PLL_CFG0, CCM_ANALOG_ARM_-

```

## Overview

```
PLL_CFG0_PLL_CLKE_SHIFT),
kCLOCK_SystemPll1Clke,
kCLOCK_SystemPll1Div2Clke,
kCLOCK_SystemPll1Div3Clke,
kCLOCK_SystemPll1Div4Clke,
kCLOCK_SystemPll1Div5Clke,
kCLOCK_SystemPll1Div6Clke,
kCLOCK_SystemPll1Div8Clke,
kCLOCK_SystemPll1Div10Clke,
kCLOCK_SystemPll1Div20Clke,
kCLOCK_SystemPll2Clke,
kCLOCK_SystemPll2Div2Clke,
kCLOCK_SystemPll2Div3Clke,
kCLOCK_SystemPll2Div4Clke,
kCLOCK_SystemPll2Div5Clke,
kCLOCK_SystemPll2Div6Clke,
kCLOCK_SystemPll2Div8Clke,
kCLOCK_SystemPll2Div10Clke,
kCLOCK_SystemPll2Div20Clke,
kCLOCK_SystemPll3Clke,
kCLOCK_VideoPll2Clke,
kCLOCK_DramPllClke = CCM_ANALOG_TUPLE(DRAM_PLL_CFG0, CCM_ANALOG_DR-
AM_PLL_CFG0_PLL_CLKE_SHIFT),
kCLOCK_OSC25MClke = CCM_ANALOG_TUPLE(OSC_MISC_CFG, CCM_ANALOG_OSC-
_MISC_CFG_OSC_25M_CLKE_SHIFT),
kCLOCK_OSC27MClke = CCM_ANALOG_TUPLE(OSC_MISC_CFG, CCM_ANALOG_OSC-
_MISC_CFG_OSC_27M_CLKE_SHIFT) }
```

*PLL clock names for clock enable/disable settings.*

- enum `clock_pll_ctrl_t`  
*ANALOG Power down override control.*
- enum `_osc_mode` {  
    `kOSC_OscMode` = 0U,  
    `kOSC_ExtMode` = 1U }
- enum `osc32_src_t` {  
    `kOSC32_Src25MDiv800` = 0U,  
    `kOSC32_SrcRTC` }
- enum `_ccm_analog_pll_ref_clk` {  
    `kANALOG_PllRefOsc25M` = 0U,  
    `kANALOG_PllRefOsc27M` = 1U,  
    `kANALOG_PllRefOscHdmiPhy27M` = 2U,  
    `kANALOG_PllRefClkPN` = 3U }

*PLL reference clock select.*

## Driver version

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

## CCM Root Clock Setting

- static void **CLOCK\_SetRootMux** (clock\_root\_control\_t rootClk, uint32\_t mux)  
*Set clock root mux.*
- static uint32\_t **CLOCK\_GetRootMux** (clock\_root\_control\_t rootClk)  
*Get clock root mux.*
- static void **CLOCK\_EnableRoot** (clock\_root\_control\_t rootClk)  
*Enable clock root.*
- static void **CLOCK\_DisableRoot** (clock\_root\_control\_t rootClk)  
*Disable clock root.*
- static bool **CLOCK\_IsRootEnabled** (clock\_root\_control\_t rootClk)  
*Check whether clock root is enabled.*
- void **CLOCK\_UpdateRoot** (clock\_root\_control\_t ccmRootClk, uint32\_t mux, uint32\_t pre, uint32\_t post)  
*Update clock root in one step, for dynamical clock switching Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.*
- void **CLOCK\_SetRootDivider** (clock\_root\_control\_t ccmRootClk, uint32\_t pre, uint32\_t post)  
*Set root clock divider Note: The PRE and POST dividers in this function are the actually divider, software will map it to register value.*
- static uint32\_t **CLOCK\_GetRootPreDivider** (clock\_root\_control\_t rootClk)  
*Get clock root PRE\_PODF.*
- static uint32\_t **CLOCK\_GetRootPostDivider** (clock\_root\_control\_t rootClk)  
*Get clock root POST\_PODF.*

## OSC setting

- void **CLOCK\_InitOSC25M** (const osc\_config\_t \*config)  
*OSC25M init.*
- void **CLOCK\_DeinitOSC25M** (void)  
*OSC25M deinit.*
- void **CLOCK\_InitOSC27M** (const osc\_config\_t \*config)  
*OSC27M init.*
- void **CLOCK\_DeinitOSC27M** (void)  
*OSC27M deinit.*
- static void **CLOCK\_SwitchOSC32Src** (osc32\_src\_t sel)  
*switch 32KHZ OSC input*

## CCM Gate Control

- static void **CLOCK\_ControlGate** (uint32\_t ccmGate, clock\_gate\_value\_t control)  
*lockrief Set PLL or CCGR gate control*
- void **CLOCK\_EnableClock** (clock\_ip\_name\_t ccmGate)  
*Enable CCGR clock gate and root clock gate for each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual.*
- void **CLOCK\_DisableClock** (clock\_ip\_name\_t ccmGate)

*Disable CCGR clock gate for the each module User should set specific gate for each module according to the description of the table of system clocks, gating and override in CCM chapter of reference manual.*

## CCM Analog PLL Operatoin Functions

- static void [CLOCK\\_PowerUpPll](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_ctrl\\_t](#) pllControl)  
*Power up PLL.*
- static void [CLOCK\\_PowerDownPll](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_ctrl\\_t](#) pllControl)  
*Power down PLL.*
- static void [CLOCK\\_SetPllBypass](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_bypass\\_ctrl\\_t](#) pllControl, bool bypass)  
*PLL bypass setting.*
- static bool [CLOCK\\_IsPllBypassed](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_bypass\\_ctrl\\_t](#) pllControl)  
*Check if PLL is bypassed.*
- static bool [CLOCK\\_IsPllLocked](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_ctrl\\_t](#) pllControl)  
*Check if PLL clock is locked.*
- static void [CLOCK\\_EnableAnalogClock](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_clke\\_t](#) pllClock)  
*Enable PLL clock.*
- static void [CLOCK\\_DisableAnalogClock](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_clke\\_t](#) pllClock)  
*Disable PLL clock.*
- static void [CLOCK\\_OverrideAnalogClke](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_clke\\_t](#) ovClock, bool override)  
*Override PLL clock output enable.*
- static void [CLOCK\\_OverridePllPd](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_ctrl\\_t](#) pdClock, bool override)  
*Override PLL power down.*
- void [CLOCK\\_InitArmPll](#) (const [ccm\\_analog\\_frac\\_pll\\_config\\_t](#) \*config)  
*Initializes the ANALOG ARM PLL.*
- void [CLOCK\\_DeinitArmPll](#) (void)  
*De-initialize the ARM PLL.*
- void [CLOCK\\_InitSysPll1](#) (const [ccm\\_analog\\_sscg\\_pll\\_config\\_t](#) \*config)  
*Initializes the ANALOG SYS PLL1.*
- void [CLOCK\\_DeinitSysPll1](#) (void)  
*De-initialize the System PLL1.*
- void [CLOCK\\_InitSysPll2](#) (const [ccm\\_analog\\_sscg\\_pll\\_config\\_t](#) \*config)  
*Initializes the ANALOG SYS PLL2.*
- void [CLOCK\\_DeinitSysPll2](#) (void)  
*De-initialize the System PLL2.*
- void [CLOCK\\_InitSysPll3](#) (const [ccm\\_analog\\_sscg\\_pll\\_config\\_t](#) \*config)  
*Initializes the ANALOG SYS PLL3.*
- void [CLOCK\\_DeinitSysPll3](#) (void)  
*De-initialize the System PLL3.*
- void [CLOCK\\_InitDramPll](#) (const [ccm\\_analog\\_sscg\\_pll\\_config\\_t](#) \*config)  
*Initializes the ANALOG DDR PLL.*
- void [CLOCK\\_DeinitDramPll](#) (void)  
*De-initialize the Dram PLL.*
- void [CLOCK\\_InitAudioPll1](#) (const [ccm\\_analog\\_frac\\_pll\\_config\\_t](#) \*config)



- *Initializes the ANALOG AUDIO PLL1.*  
void [CLOCK\\_DeinitAudioPll1](#) (void)
- *De-initialize the Audio PLL1.*  
void [CLOCK\\_InitAudioPll2](#) (const [ccm\\_analog\\_frac\\_pll\\_config\\_t](#) \*config)
- *Initializes the ANALOG AUDIO PLL2.*  
void [CLOCK\\_DeinitAudioPll2](#) (void)
- *De-initialize the Audio PLL2.*  
void [CLOCK\\_InitVideoPll1](#) (const [ccm\\_analog\\_frac\\_pll\\_config\\_t](#) \*config)
- *Initializes the ANALOG VIDEO PLL1.*  
void [CLOCK\\_DeinitVideoPll1](#) (void)
- *De-initialize the Video PLL1.*  
void [CLOCK\\_InitVideoPll2](#) (const [ccm\\_analog\\_sscg\\_pll\\_config\\_t](#) \*config)
- *Initializes the ANALOG VIDEO PLL2.*  
void [CLOCK\\_DeinitVideoPll2](#) (void)
- *De-initialize the Video PLL2.*  
void [CLOCK\\_InitSSCGPll](#) (CCM\_ANALOG\_Type \*base, const [ccm\\_analog\\_sscg\\_pll\\_config\\_t](#) \*config, [clock\\_pll\\_ctrl\\_t](#) type)
- *Initializes the ANALOG SSCG PLL.*  
uint32\_t [CLOCK\\_GetSSCGPllFreq](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_ctrl\\_t](#) type, uint32\_t refClkFreq, bool pll1Bypass)
- *Get the ANALOG SSCG PLL clock frequency.*  
void [CLOCK\\_InitFracPll](#) (CCM\_ANALOG\_Type \*base, const [ccm\\_analog\\_frac\\_pll\\_config\\_t](#) \*config, [clock\\_pll\\_ctrl\\_t](#) type)
- *Initializes the ANALOG Fractional PLL.*  
uint32\_t [CLOCK\\_GetFracPllFreq](#) (CCM\_ANALOG\_Type \*base, [clock\\_pll\\_ctrl\\_t](#) type, uint32\_t refClkFreq)
- *Gets the ANALOG Fractional PLL clock frequency.*  
uint32\_t [CLOCK\\_GetPllFreq](#) ([clock\\_pll\\_ctrl\\_t](#) pll)
- *Gets PLL clock frequency.*  
uint32\_t [CLOCK\\_GetPllRefClkFreq](#) ([clock\\_pll\\_ctrl\\_t](#) ctrl)
- *Gets PLL reference clock frequency.*

## CCM Get frequency

- uint32\_t [CLOCK\\_GetFreq](#) ([clock\\_name\\_t](#) clockName)  
*Gets the clock frequency for a specific clock name.*
- uint32\_t [CLOCK\\_GetCoreM4Freq](#) (void)  
*Get the CCM Cortex M4 core frequency.*
- uint32\_t [CLOCK\\_GetAxiFreq](#) (void)  
*Get the CCM Axi bus frequency.*
- uint32\_t [CLOCK\\_GetAhbFreq](#) (void)  
*Get the CCM Ahb bus frequency.*

## 5.2 Data Structure Documentation

### 5.2.1 struct osc\_config\_t

#### Data Fields

- uint8\_t [oscMode](#)

- ext or osc mode*
- uint8\_t [oscDiv](#)  
*osc divider*

### 5.2.2 struct ccm\_analog\_frac\_pll\_config\_t

Note: all the dividers in this configuration structure are the actually divider, software will map it to register value

#### Data Fields

- uint8\_t [refSel](#)  
*pll reference clock sel*
- uint8\_t [refDiv](#)  
*A 6bit divider to make sure the REF must be within the range 10MHZ~300MHZ.*
- uint32\_t [fractionDiv](#)  
*Include fraction divider(divider:1:2<sup>24</sup>) output clock range is 2000MHZ-4000MHZ.*
- uint8\_t [outDiv](#)  
*output clock divide, output clock range is 30MHZ to 2000MHZ, must be a even value*

### 5.2.3 struct ccm\_analog\_sscg\_pll\_config\_t

Note: all the dividers in this configuration structure are the actually divider, software will map it to register value

#### Data Fields

- uint8\_t [refSel](#)  
*pll reference clock sel*
- uint8\_t [refDiv1](#)  
*A 3bit divider to make sure the REF must be within the range 25MHZ~235MHZ ,post\_divide REF must be within the range 25MHZ~54MHZ.*
- uint8\_t [refDiv2](#)  
*A 6bit divider to make sure the post\_divide REF must be within the range 54MHZ~75MHZ.*
- uint32\_t [loopDivider1](#)  
*A 6bit internal PLL1 feedback clock divider, output clock range must be within the range 1600MHZ-2400-MHZ.*
- uint32\_t [loopDivider2](#)  
*A 6bit internal PLL2 feedback clock divider, output clock range must be within the range 1200MHZ-2400-MHZ.*
- uint8\_t [outDiv](#)  
*A 6bit output clock divide, output clock range is 20MHZ to 1200MHZ.*

## 5.3 Macro Definition Documentation

### 5.3.1 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 5.3.2 #define ECSPI\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Ecspi1, kCLOCK_Ecspi2, \
    kCLOCK_Ecspi3, \
}
```

### 5.3.3 #define GPIO\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Gpio1, kCLOCK_Gpio2, \
    kCLOCK_Gpio3, kCLOCK_Gpio4, kCLOCK_Gpio5, \
}
```

### 5.3.4 #define GPT\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Gpt1, kCLOCK_Gpt2, \
    kCLOCK_Gpt3, kCLOCK_Gpt4, kCLOCK_Gpt5, \
    kCLOCK_Gpt6, \
}
```

### 5.3.5 #define I2C\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_I2c1, kCLOCK_I2c2, \
    kCLOCK_I2c3, kCLOCK_I2c4, \
}
```

## Macro Definition Documentation

### 5.3.6 #define IOMUX\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Iomuxc0, kCLOCK_Iomuxc1, kCLOCK_Iomuxc2, kCLOCK_Iomuxc3, kCLOCK_Iomuxc4, \
}
```

### 5.3.7 #define PWM\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Pwm1, kCLOCK_Pwm2, \
    kCLOCK_Pwm3, kCLOCK_Pwm4, \
}
```

### 5.3.8 #define RDC\_CLOCKS

Value:

```
{
    kCLOCK_Rdc, \
}
```

### 5.3.9 #define SAI\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Sai1, kCLOCK_Sai2, \
    kCLOCK_Sai3, kCLOCK_Sai4, kCLOCK_Sai5, \
    kCLOCK_Sai6, \
}
```

### 5.3.10 #define RDC\_SEMA42\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Sema42_1, kCLOCK_Sema42_2 \
}
```

### 5.3.11 #define UART\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Uart1, kCLOCK_Uart2,
    kCLOCK_Uart3, kCLOCK_Uart4, \
}
```

### 5.3.12 #define USDHC\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Usdhc1, kCLOCK_Usdhc2 \
}
```

### 5.3.13 #define WDOG\_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Wdog1, kCLOCK_Wdog2,
    kCLOCK_Wdog3 \
}
```

### 5.3.14 #define TMU\_CLOCKS

Value:

```
{
    kCLOCK_TempSensor, \
}
```

### 5.3.15 #define SDMA\_CLOCKS

Value:

```
{
    kCLOCK_Sdma1, kCLOCK_Sdma2 \
}
```

## Enumeration Type Documentation

### 5.3.16 #define MU\_CLOCKS

Value:

```
{  
    \kCLOCK_Mu \  
}
```

### 5.3.17 #define QSPI\_CLOCKS

Value:

```
{  
    \kCLOCK_Qspi \  
}
```

### 5.3.18 #define kCLOCK\_CoreSysClk kCLOCK\_CoreM4Clk

### 5.3.19 #define CLOCK\_GetCoreSysClkFreq CLOCK\_GetCoreM4Freq

## 5.4 Enumeration Type Documentation

### 5.4.1 enum clock\_name\_t

Enumerator

*kCLOCK\_CoreM4Clk* ARM M4 Core clock.  
*kCLOCK\_AxiClk* Main AXI bus clock.  
*kCLOCK\_AhbClk* AHB bus clock.  
*kCLOCK\_IpgClk* IPG bus clock.

### 5.4.2 enum clock\_ip\_name\_t

Enumerator

*kCLOCK\_Debug* DEBUG Clock Gate.  
*kCLOCK\_Dram* DRAM Clock Gate.  
*kCLOCK\_Ecspi1* ECSPI1 Clock Gate.  
*kCLOCK\_Ecspi2* ECSPI2 Clock Gate.  
*kCLOCK\_Ecspi3* ECSPI3 Clock Gate.  
*kCLOCK\_Gpio1* GPIO1 Clock Gate.  
*kCLOCK\_Gpio2* GPIO2 Clock Gate.

***kCLOCK\_Gpio3*** GPIO3 Clock Gate.  
***kCLOCK\_Gpio4*** GPIO4 Clock Gate.  
***kCLOCK\_Gpio5*** GPIO5 Clock Gate.  
***kCLOCK\_Gpt1*** GPT1 Clock Gate.  
***kCLOCK\_Gpt2*** GPT2 Clock Gate.  
***kCLOCK\_Gpt3*** GPT3 Clock Gate.  
***kCLOCK\_Gpt4*** GPT4 Clock Gate.  
***kCLOCK\_Gpt5*** GPT5 Clock Gate.  
***kCLOCK\_Gpt6*** GPT6 Clock Gate.  
***kCLOCK\_I2c1*** I2C1 Clock Gate.  
***kCLOCK\_I2c2*** I2C2 Clock Gate.  
***kCLOCK\_I2c3*** I2C3 Clock Gate.  
***kCLOCK\_I2c4*** I2C4 Clock Gate.  
***kCLOCK\_Iomux0*** IOMUX Clock Gate.  
***kCLOCK\_Iomux1*** IOMUX Clock Gate.  
***kCLOCK\_Iomux2*** IOMUX Clock Gate.  
***kCLOCK\_Iomux3*** IOMUX Clock Gate.  
***kCLOCK\_Iomux4*** IOMUX Clock Gate.  
***kCLOCK\_M4*** M4 Clock Gate.  
***kCLOCK\_Mu*** MU Clock Gate.  
***kCLOCK\_Ocram*** OCRAM Clock Gate.  
***kCLOCK\_OcramS*** OCRAM S Clock Gate.  
***kCLOCK\_Pwm1*** PWM1 Clock Gate.  
***kCLOCK\_Pwm2*** PWM2 Clock Gate.  
***kCLOCK\_Pwm3*** PWM3 Clock Gate.  
***kCLOCK\_Pwm4*** PWM4 Clock Gate.  
***kCLOCK\_Qspi*** QSPI Clock Gate.  
***kCLOCK\_Rdc*** RDC Clock Gate.  
***kCLOCK\_Sai1*** SAI1 Clock Gate.  
***kCLOCK\_Sai2*** SAI2 Clock Gate.  
***kCLOCK\_Sai3*** SAI3 Clock Gate.  
***kCLOCK\_Sai4*** SAI4 Clock Gate.  
***kCLOCK\_Sai5*** SAI5 Clock Gate.  
***kCLOCK\_Sai6*** SAI6 Clock Gate.  
***kCLOCK\_Sdma1*** SDMA1 Clock Gate.  
***kCLOCK\_Sdma2*** SDMA2 Clock Gate.  
***kCLOCK\_Sec\_Debug*** SEC\_DEBUG Clock Gate.  
***kCLOCK\_Sema42\_1*** RDC SEMA42 Clock Gate.  
***kCLOCK\_Sema42\_2*** RDC SEMA42 Clock Gate.  
***kCLOCK\_Sim\_display*** SIM\_Display Clock Gate.  
***kCLOCK\_Sim\_m*** SIM\_M Clock Gate.  
***kCLOCK\_Sim\_main*** SIM\_MAIN Clock Gate.  
***kCLOCK\_Sim\_s*** SIM\_S Clock Gate.  
***kCLOCK\_Sim\_wakeup*** SIM\_WAKEUP Clock Gate.  
***kCLOCK\_Uart1*** UART1 Clock Gate.

## Enumeration Type Documentation

***kCLOCK\_Uart2*** UART2 Clock Gate.  
***kCLOCK\_Uart3*** UART3 Clock Gate.  
***kCLOCK\_Uart4*** UART4 Clock Gate.  
***kCLOCK\_Wdog1*** WDOG1 Clock Gate.  
***kCLOCK\_Wdog2*** WDOG2 Clock Gate.  
***kCLOCK\_Wdog3*** WDOG3 Clock Gate.  
***kCLOCK\_TempSensor*** TempSensor Clock Gate.

### 5.4.3 enum clock\_root\_control\_t

Enumerator

***kCLOCK\_RootM4*** ARM Cortex-M4 Clock control name.  
***kCLOCK\_RootAxi*** AXI Clock control name.  
***kCLOCK\_RootNoc*** NOC Clock control name.  
***kCLOCK\_RootAhb*** AHB Clock control name.  
***kCLOCK\_RootIpg*** IPG Clock control name.  
***kCLOCK\_RootDramAlt*** DRAM ALT Clock control name.  
***kCLOCK\_RootSai1*** SAI1 Clock control name.  
***kCLOCK\_RootSai2*** SAI2 Clock control name.  
***kCLOCK\_RootSai3*** SAI3 Clock control name.  
***kCLOCK\_RootSai4*** SAI4 Clock control name.  
***kCLOCK\_RootSai5*** SAI5 Clock control name.  
***kCLOCK\_RootSai6*** SAI6 Clock control name.  
***kCLOCK\_RootQspi*** QSPI Clock control name.  
***kCLOCK\_RootI2c1*** I2C1 Clock control name.  
***kCLOCK\_RootI2c2*** I2C2 Clock control name.  
***kCLOCK\_RootI2c3*** I2C3 Clock control name.  
***kCLOCK\_RootI2c4*** I2C4 Clock control name.  
***kCLOCK\_RootUart1*** UART1 Clock control name.  
***kCLOCK\_RootUart2*** UART2 Clock control name.  
***kCLOCK\_RootUart3*** UART3 Clock control name.  
***kCLOCK\_RootUart4*** UART4 Clock control name.  
***kCLOCK\_RootEcspi1*** ECSPI1 Clock control name.  
***kCLOCK\_RootEcspi2*** ECSPI2 Clock control name.  
***kCLOCK\_RootEcspi3*** ECSPI3 Clock control name.  
***kCLOCK\_RootPwm1*** PWM1 Clock control name.  
***kCLOCK\_RootPwm2*** PWM2 Clock control name.  
***kCLOCK\_RootPwm3*** PWM3 Clock control name.  
***kCLOCK\_RootPwm4*** PWM4 Clock control name.  
***kCLOCK\_RootGpt1*** GPT1 Clock control name.  
***kCLOCK\_RootGpt2*** GPT2 Clock control name.  
***kCLOCK\_RootGpt3*** GPT3 Clock control name.  
***kCLOCK\_RootGpt4*** GPT4 Clock control name.



***kCLOCK\_RootGpt5*** GPT5 Clock control name.  
***kCLOCK\_RootGpt6*** GPT6 Clock control name.  
***kCLOCK\_RootWdog*** WDOG Clock control name.

#### 5.4.4 enum clock\_rootmux\_m4\_clk\_sel\_t

Enumerator

***kCLOCK\_M4RootmuxOsc25m*** ARM Cortex-M4 Clock from OSC 25M.  
***kCLOCK\_M4RootmuxSysPll2Div5*** ARM Cortex-M4 Clock from SYSTEM PLL2 divided by 5.  
***kCLOCK\_M4RootmuxSysPll2Div4*** ARM Cortex-M4 Clock from SYSTEM PLL2 divided by 4.  
***kCLOCK\_M4RootmuxSysPll1Div3*** ARM Cortex-M4 Clock from SYSTEM PLL1 divided by 3.  
***kCLOCK\_M4RootmuxSysPll1*** ARM Cortex-M4 Clock from SYSTEM PLL1.  
***kCLOCK\_M4RootmuxAudioPll1*** ARM Cortex-M4 Clock from AUDIO PLL1.  
***kCLOCK\_M4RootmuxVideoPll1*** ARM Cortex-M4 Clock from VIDEO PLL1.  
***kCLOCK\_M4RootmuxSysPll3*** ARM Cortex-M4 Clock from SYSTEM PLL3.

#### 5.4.5 enum clock\_rootmux\_axi\_clk\_sel\_t

Enumerator

***kCLOCK\_AxiRootmuxOsc25m*** ARM AXI Clock from OSC 25M.  
***kCLOCK\_AxiRootmuxSysPll2Div3*** ARM AXI Clock from SYSTEM PLL2 divided by 3.  
***kCLOCK\_AxiRootmuxSysPll1*** ARM AXI Clock from SYSTEM PLL1.  
***kCLOCK\_AxiRootmuxSysPll2Div4*** ARM AXI Clock from SYSTEM PLL2 divided by 4.  
***kCLOCK\_AxiRootmuxSysPll2*** ARM AXI Clock from SYSTEM PLL2.  
***kCLOCK\_AxiRootmuxAudioPll1*** ARM AXI Clock from AUDIO PLL1.  
***kCLOCK\_AxiRootmuxVideoPll1*** ARM AXI Clock from VIDEO PLL1.  
***kCLOCK\_AxiRootmuxSysPll1Div8*** ARM AXI Clock from SYSTEM PLL1 divided by 8.

#### 5.4.6 enum clock\_rootmux\_ahb\_clk\_sel\_t

Enumerator

***kCLOCK\_AhbRootmuxOsc25m*** ARM AHB Clock from OSC 25M.  
***kCLOCK\_AhbRootmuxSysPll1Div6*** ARM AHB Clock from SYSTEM PLL1 divided by 6.  
***kCLOCK\_AhbRootmuxSysPll1*** ARM AHB Clock from SYSTEM PLL1.  
***kCLOCK\_AhbRootmuxSysPll1Div2*** ARM AHB Clock from SYSTEM PLL1 divided by 2.  
***kCLOCK\_AhbRootmuxSysPll2Div8*** ARM AHB Clock from SYSTEM PLL2 divided by 8.  
***kCLOCK\_AhbRootmuxSysPll3*** ARM AHB Clock from SYSTEM PLL3.  
***kCLOCK\_AhbRootmuxAudioPll1*** ARM AHB Clock from AUDIO PLL1.  
***kCLOCK\_AhbRootmuxVideoPll1*** ARM AHB Clock from VIDEO PLL1.

### 5.4.7 enum clock\_rootmux\_qspi\_clk\_sel\_t

Enumerator

*kCLOCK\_QspiRootmuxOsc25m* ARM QSPI Clock from OSC 25M.  
*kCLOCK\_QspiRootmuxSysPll1Div2* ARM QSPI Clock from SYSTEM PLL1 divided by 2.  
*kCLOCK\_QspiRootmuxSysPll1* ARM QSPI Clock from SYSTEM PLL1.  
*kCLOCK\_QspiRootmuxSysPll2Div2* ARM QSPI Clock from SYSTEM PLL2 divided by 2.  
*kCLOCK\_QspiRootmuxAudioPll2* ARM QSPI Clock from AUDIO PLL2.  
*kCLOCK\_QspiRootmuxSysPll1Div3* ARM QSPI Clock from SYSTEM PLL1 divided by 3.  
*kCLOCK\_QspiRootmuxSysPll3* ARM QSPI Clock from SYSTEM PLL3.  
*kCLOCK\_QspiRootmuxSysPll1Div8* ARM QSPI Clock from SYSTEM PLL1 divided by 8.

### 5.4.8 enum clock\_rootmux\_ecspi\_clk\_sel\_t

Enumerator

*kCLOCK\_EcspiRootmuxOsc25m* ECSPI Clock from OSC 25M.  
*kCLOCK\_EcspiRootmuxSysPll2Div5* ECSPI Clock from SYSTEM PLL2 divided by 5.  
*kCLOCK\_EcspiRootmuxSysPll1Div20* ECSPI Clock from SYSTEM PLL1 divided by 20.  
*kCLOCK\_EcspiRootmuxSysPll1Div5* ECSPI Clock from SYSTEM PLL1 divided by 5.  
*kCLOCK\_EcspiRootmuxSysPll1* ECSPI Clock from SYSTEM PLL1.  
*kCLOCK\_EcspiRootmuxSysPll3* ECSPI Clock from SYSTEM PLL3.  
*kCLOCK\_EcspiRootmuxSysPll2Div4* ECSPI Clock from SYSTEM PLL2 divided by 4.  
*kCLOCK\_EcspiRootmuxAudioPll2* ECSPI Clock from AUDIO PLL2.

### 5.4.9 enum clock\_rootmux\_i2c\_clk\_sel\_t

Enumerator

*kCLOCK\_I2cRootmuxOsc25m* I2C Clock from OSC 25M.  
*kCLOCK\_I2cRootmuxSysPll1Div5* I2C Clock from SYSTEM PLL1 divided by 5.  
*kCLOCK\_I2cRootmuxSysPll2Div20* I2C Clock from SYSTEM PLL2 divided by 20.  
*kCLOCK\_I2cRootmuxSysPll3* I2C Clock from SYSTEM PLL3 .  
*kCLOCK\_I2cRootmuxAudioPll1* I2C Clock from AUDIO PLL1.  
*kCLOCK\_I2cRootmuxVideoPll1* I2C Clock from VIDEO PLL1.  
*kCLOCK\_I2cRootmuxAudioPll2* I2C Clock from AUDIO PLL2.  
*kCLOCK\_I2cRootmuxSysPll1Div6* I2C Clock from SYSTEM PLL1 divided by 6.

#### 5.4.10 enum clock\_rootmux\_uart\_clk\_sel\_t

Enumerator

*kCLOCK\_UartRootmuxOsc25m* UART Clock from OSC 25M.  
*kCLOCK\_UartRootmuxSysPll1Div10* UART Clock from SYSTEM PLL1 divided by 10.  
*kCLOCK\_UartRootmuxSysPll2Div5* UART Clock from SYSTEM PLL2 divided by 5.  
*kCLOCK\_UartRootmuxSysPll2Div10* UART Clock from SYSTEM PLL2 divided by 10.  
*kCLOCK\_UartRootmuxSysPll3* UART Clock from SYSTEM PLL3.  
*kCLOCK\_UartRootmuxExtClk2* UART Clock from External Clock 2.  
*kCLOCK\_UartRootmuxExtClk34* UART Clock from External Clock 3, External Clock 4.  
*kCLOCK\_UartRootmuxAudioPll2* UART Clock from Audio PLL2.

#### 5.4.11 enum clock\_rootmux\_gpt\_t

Enumerator

*kCLOCK\_GptRootmuxOsc25m* GPT Clock from OSC 25M.  
*kCLOCK\_GptRootmuxSystemPll2Div10* GPT Clock from SYSTEM PLL2 divided by 10.  
*kCLOCK\_GptRootmuxSysPll1Div2* GPT Clock from SYSTEM PLL1 divided by 2.  
*kCLOCK\_GptRootmuxSysPll1Div20* GPT Clock from SYSTEM PLL1 divided by 20.  
*kCLOCK\_GptRootmuxVideoPll1* GPT Clock from VIDEO PLL1.  
*kCLOCK\_GptRootmuxSystemPll1Div10* GPT Clock from SYSTEM PLL1 divided by 10.  
*kCLOCK\_GptRootmuxAudioPll1* GPT Clock from AUDIO PLL1.  
*kCLOCK\_GptRootmuxExtClk123* GPT Clock from External Clock1, External Clock2, External Clock3.

#### 5.4.12 enum clock\_rootmux\_wdog\_clk\_sel\_t

Enumerator

*kCLOCK\_WdogRootmuxOsc25m* WDOG Clock from OSC 25M.  
*kCLOCK\_WdogRootmuxSysPll1Div6* WDOG Clock from SYSTEM PLL1 divided by 6.  
*kCLOCK\_WdogRootmuxSysPll1Div5* WDOG Clock from SYSTEM PLL1 divided by 5.  
*kCLOCK\_WdogRootmuxVpuPll* WDOG Clock from VPU DLL.  
*kCLOCK\_WdogRootmuxSystemPll2Div8* WDOG Clock from SYSTEM PLL2 divided by 8.  
*kCLOCK\_WdogRootmuxSystemPll3* WDOG Clock from SYSTEM PLL3.  
*kCLOCK\_WdogRootmuxSystemPll1Div10* WDOG Clock from SYSTEM PLL1 divided by 10.  
*kCLOCK\_WdogRootmuxSystemPll2Div6* WDOG Clock from SYSTEM PLL2 divided by 6.

### 5.4.13 enum clock\_rootmux\_Pwm\_clk\_sel\_t

Enumerator

*kCLOCK\_PwmRootmuxOsc25m* PWM Clock from OSC 25M.  
*kCLOCK\_PwmRootmuxSysPll2Div10* PWM Clock from SYSTEM PLL2 divided by 10.  
*kCLOCK\_PwmRootmuxSysPll1Div5* PWM Clock from SYSTEM PLL1 divided by 5.  
*kCLOCK\_PwmRootmuxSysPll1Div20* PWM Clock from SYSTEM PLL1 divided by 20.  
*kCLOCK\_PwmRootmuxSystemPll3* PWM Clock from SYSTEM PLL3.  
*kCLOCK\_PwmRootmuxExtClk12* PWM Clock from External Clock1, External Clock2.  
*kCLOCK\_PwmRootmuxSystemPll1Div10* PWM Clock from SYSTEM PLL1 divided by 10.  
*kCLOCK\_PwmRootmuxVideoPll1* PWM Clock from VIDEO PLL1.

### 5.4.14 enum clock\_rootmux\_sai\_clk\_sel\_t

Enumerator

*kCLOCK\_SaiRootmuxOsc25m* SAI Clock from OSC 25M.  
*kCLOCK\_SaiRootmuxAudioPll1* SAI Clock from AUDIO PLL1.  
*kCLOCK\_SaiRootmuxAudioPll2* SAI Clock from AUDIO PLL2.  
*kCLOCK\_SaiRootmuxVideoPll1* SAI Clock from VIDEO PLL1.  
*kCLOCK\_SaiRootmuxSysPll1Div6* SAI Clock from SYSTEM PLL1 divided by 6.  
*kCLOCK\_SaiRootmuxOsc27m* SAI Clock from OSC 27M.  
*kCLOCK\_SaiRootmuxExtClk123* SAI Clock from External Clock1, External Clock2, External Clock3.  
*kCLOCK\_SaiRootmuxExtClk234* SAI Clock from External Clock2, External Clock3, External Clock4.

### 5.4.15 enum clock\_rootmux\_noc\_clk\_sel\_t

Enumerator

*kCLOCK\_NocRootmuxOsc25m* NOC Clock from OSC 25M.  
*kCLOCK\_NocRootmuxSysPll1* NOC Clock from SYSTEM PLL1.  
*kCLOCK\_NocRootmuxSysPll3* NOC Clock from SYSTEM PLL3.  
*kCLOCK\_NocRootmuxSysPll2* NOC Clock from SYSTEM PLL2.  
*kCLOCK\_NocRootmuxSysPll2Div2* NOC Clock from SYSTEM PLL2 divided by 2.  
*kCLOCK\_NocRootmuxAudioPll1* NOC Clock from AUDIO PLL1.  
*kCLOCK\_NocRootmuxVideoPll1* NOC Clock from VIDEO PLL1.  
*kCLOCK\_NocRootmuxAudioPll2* NOC Clock from AUDIO PLL2.

### 5.4.16 enum clock\_pll\_gate\_t

Enumerator

***kCLOCK\_ArmPllGate*** ARM PLL Gate.  
***kCLOCK\_GpuPllGate*** GPU PLL Gate.  
***kCLOCK\_VpuPllGate*** VPU PLL Gate.  
***kCLOCK\_DramPllGate*** DRAM PLL1 Gate.  
***kCLOCK\_SysPll1Gate*** SYSTEM PLL1 Gate.  
***kCLOCK\_SysPll1Div2Gate*** SYSTEM PLL1 Div2 Gate.  
***kCLOCK\_SysPll1Div3Gate*** SYSTEM PLL1 Div3 Gate.  
***kCLOCK\_SysPll1Div4Gate*** SYSTEM PLL1 Div4 Gate.  
***kCLOCK\_SysPll1Div5Gate*** SYSTEM PLL1 Div5 Gate.  
***kCLOCK\_SysPll1Div6Gate*** SYSTEM PLL1 Div6 Gate.  
***kCLOCK\_SysPll1Div8Gate*** SYSTEM PLL1 Div8 Gate.  
***kCLOCK\_SysPll1Div10Gate*** SYSTEM PLL1 Div10 Gate.  
***kCLOCK\_SysPll1Div20Gate*** SYSTEM PLL1 Div20 Gate.  
***kCLOCK\_SysPll2Gate*** SYSTEM PLL2 Gate.  
***kCLOCK\_SysPll2Div2Gate*** SYSTEM PLL2 Div2 Gate.  
***kCLOCK\_SysPll2Div3Gate*** SYSTEM PLL2 Div3 Gate.  
***kCLOCK\_SysPll2Div4Gate*** SYSTEM PLL2 Div4 Gate.  
***kCLOCK\_SysPll2Div5Gate*** SYSTEM PLL2 Div5 Gate.  
***kCLOCK\_SysPll2Div6Gate*** SYSTEM PLL2 Div6 Gate.  
***kCLOCK\_SysPll2Div8Gate*** SYSTEM PLL2 Div8 Gate.  
***kCLOCK\_SysPll2Div10Gate*** SYSTEM PLL2 Div10 Gate.  
***kCLOCK\_SysPll2Div20Gate*** SYSTEM PLL2 Div20 Gate.  
***kCLOCK\_SysPll3Gate*** SYSTEM PLL3 Gate.  
***kCLOCK\_AudioPll1Gate*** AUDIO PLL1 Gate.  
***kCLOCK\_AudioPll2Gate*** AUDIO PLL2 Gate.  
***kCLOCK\_VideoPll1Gate*** VIDEO PLL1 Gate.  
***kCLOCK\_VideoPll2Gate*** VIDEO PLL2 Gate.

### 5.4.17 enum clock\_gate\_value\_t

Enumerator

***kCLOCK\_ClockNotNeeded*** Clock always disabled.  
***kCLOCK\_ClockNeededRun*** Clock enabled when CPU is running.  
***kCLOCK\_ClockNeededRunWait*** Clock enabled when CPU is running or in WAIT mode.  
***kCLOCK\_ClockNeededAll*** Clock always enabled.

### 5.4.18 enum clock\_pll\_bypass\_ctrl\_t

These constants define the PLL control names for PLL bypass.

- 0:15: REG offset to CCM\_ANALOG\_BASE in bytes.
- 16:20: bypass bit shift.

Enumerator

***kCLOCK\_AudioPll1BypassCtrl*** CCM Audio PLL1 bypass Control.  
***kCLOCK\_AudioPll2BypassCtrl*** CCM Audio PLL2 bypass Control.  
***kCLOCK\_VideoPll1BypassCtrl*** CCM Video Pll1 bypass Control.  
***kCLOCK\_GpuPllPwrBypassCtrl*** CCM Gpu PLL bypass Control.  
***kCLOCK\_VpuPllPwrBypassCtrl*** CCM Vpu PLL bypass Control.  
***kCLOCK\_ArmPllPwrBypassCtrl*** CCM Arm PLL bypass Control.  
***kCLOCK\_SysPll1InternalPll1BypassCtrl*** CCM System PLL1 internal pll1 bypass Control.  
***kCLOCK\_SysPll1InternalPll2BypassCtrl*** CCM System PLL1 internal pll2 bypass Control.  
***kCLOCK\_SysPll2InternalPll1BypassCtrl*** CCM Analog System PLL1 internal pll1 bypass Control.  
  
***kCLOCK\_SysPll2InternalPll2BypassCtrl*** CCM Analog VIDEO System PLL1 internal pll1 bypass Control.  
***kCLOCK\_SysPll3InternalPll1BypassCtrl*** CCM Analog VIDEO PLL bypass Control.  
***kCLOCK\_SysPll3InternalPll2BypassCtrl*** CCM Analog VIDEO PLL bypass Control.  
***kCLOCK\_VideoPll2InternalPll1BypassCtrl*** CCM Analog 480M PLL bypass Control.  
***kCLOCK\_VideoPll2InternalPll2BypassCtrl*** CCM Analog 480M PLL bypass Control.  
***kCLOCK\_DramPllInternalPll1BypassCtrl*** CCM Analog 480M PLL bypass Control.  
***kCLOCK\_DramPllInternalPll2BypassCtrl*** CCM Analog 480M PLL bypass Control.

### 5.4.19 enum clock\_pll\_clke\_t

These constants define the PLL clock names for PLL clock enable/disable operations.

- 0:15: REG offset to CCM\_ANALOG\_BASE in bytes.
- 16:20: Clock enable bit shift.

Enumerator

***kCLOCK\_AudioPll1Clke*** Audio pll1 clke.  
***kCLOCK\_AudioPll2Clke*** Audio pll2 clke.  
***kCLOCK\_VideoPll1Clke*** Video pll1 clke.  
***kCLOCK\_GpuPllClke*** Gpu pll clke.  
***kCLOCK\_VpuPllClke*** Vpu pll clke.  
***kCLOCK\_ArmPllClke*** Arm pll clke.  
***kCLOCK\_SystemPll1Clke*** System pll1 clke.  
***kCLOCK\_SystemPll1Div2Clke*** System pll1 Div2 clke.

*kCLOCK\_SystemPll1Div3Clke* System pll1 Div3 clke.  
*kCLOCK\_SystemPll1Div4Clke* System pll1 Div4 clke.  
*kCLOCK\_SystemPll1Div5Clke* System pll1 Div5 clke.  
*kCLOCK\_SystemPll1Div6Clke* System pll1 Div6 clke.  
*kCLOCK\_SystemPll1Div8Clke* System pll1 Div8 clke.  
*kCLOCK\_SystemPll1Div10Clke* System pll1 Div10 clke.  
*kCLOCK\_SystemPll1Div20Clke* System pll1 Div20 clke.  
*kCLOCK\_SystemPll2Clke* System pll2 clke.  
*kCLOCK\_SystemPll2Div2Clke* System pll2 Div2 clke.  
*kCLOCK\_SystemPll2Div3Clke* System pll2 Div3 clke.  
*kCLOCK\_SystemPll2Div4Clke* System pll2 Div4 clke.  
*kCLOCK\_SystemPll2Div5Clke* System pll2 Div5 clke.  
*kCLOCK\_SystemPll2Div6Clke* System pll2 Div6 clke.  
*kCLOCK\_SystemPll2Div8Clke* System pll2 Div8 clke.  
*kCLOCK\_SystemPll2Div10Clke* System pll2 Div10 clke.  
*kCLOCK\_SystemPll2Div20Clke* System pll2 Div20 clke.  
*kCLOCK\_SystemPll3Clke* System pll3 clke.  
*kCLOCK\_VideoPll2Clke* Video pll2 clke.  
*kCLOCK\_DramPllClke* Dram pll clke.  
*kCLOCK\_OSC25MClke* OSC25M clke.  
*kCLOCK\_OSC27MClke* OSC27M clke.

#### 5.4.20 enum \_osc\_mode

Enumerator

*kOSC\_OscMode* OSC oscillator mode.  
*kOSC\_ExtMode* OSC external mode.

#### 5.4.21 enum osc32\_src\_t

Enumerator

*kOSC32\_Src25MDiv800* source from 25M divide 800  
*kOSC32\_SrcRTC* source from RTC

#### 5.4.22 enum \_ccm\_analog\_pll\_ref\_clk

Enumerator

*kANALOG\_PllRefOsc25M* reference OSC 25M

## Function Documentation

*kANALOG\_PllRefOsc27M* reference OSC 27M

*kANALOG\_PllRefOscHdmiPhy27M* reference HDMI PHY 27M

*kANALOG\_PllRefClkPN* reference CLK\_P\_N

## 5.5 Function Documentation

### 5.5.1 static void CLOCK\_SetRootMux ( clock\_root\_control\_t *rootClk*, uint32\_t *mux* ) [inline], [static]

User maybe need to set more than one mux ROOT according to the clock tree description in the reference manual.

Parameters

<i>rootClk</i>	Root clock control (see <a href="#">clock_root_control_t</a> enumeration).
<i>mux</i>	Root mux value (see <a href="#">_ccm_rootmux_xxx</a> enumeration).

### 5.5.2 static uint32\_t CLOCK\_GetRootMux ( clock\_root\_control\_t *rootClk* ) [inline], [static]

In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

<i>rootClk</i>	Root clock control (see <a href="#">clock_root_control_t</a> enumeration).
----------------	--

Returns

Root mux value (see [\\_ccm\\_rootmux\\_xxx](#) enumeration).

### 5.5.3 static void CLOCK\_EnableRoot ( clock\_root\_control\_t *rootClk* ) [inline], [static]

Parameters



<i>base</i>	CCM base pointer.
<i>rootClk</i>	Root clock control (see <a href="#">clock_root_control_t</a> enumeration)

#### 5.5.4 static void **CLOCK\_DisableRoot** ( [clock\\_root\\_control\\_t](#) *rootClk* ) [**inline**], [**static**]

Parameters

<i>base</i>	CCM base pointer.
<i>rootClk</i>	Root control (see <a href="#">clock_root_control_t</a> enumeration)

#### 5.5.5 static bool **CLOCK\_IsRootEnabled** ( [clock\\_root\\_control\\_t](#) *rootClk* ) [**inline**], [**static**]

Parameters

<i>base</i>	CCM base pointer.
<i>rootClk</i>	Root control (see <a href="#">clock_root_control_t</a> enumeration)

Returns

CCM root enabled or not.

- true: Clock root is enabled.
- false: Clock root is disabled.

#### 5.5.6 void **CLOCK\_UpdateRoot** ( [clock\\_root\\_control\\_t](#) *ccmRootClk*, [uint32\\_t](#) *mux*, [uint32\\_t](#) *pre*, [uint32\\_t](#) *post* )

Parameters

<i>ccmRootClk</i>	Root control (see <a href="#">clock_root_control_t</a> enumeration)
<i>root</i>	mux value (see <a href="#">_ccm_rootmux_xxx</a> enumeration)

## Function Documentation

<i>pre</i>	Pre divider value (0-7, divider=n+1)
<i>post</i>	Post divider value (0-63, divider=n+1)

### 5.5.7 void CLOCK\_SetRootDivider ( clock\_root\_control\_t *ccmRootClk*, uint32\_t *pre*, uint32\_t *post* )

Parameters

<i>ccmRootClk</i>	Root control (see <a href="#">clock_root_control_t</a> enumeration)
<i>pre</i>	Pre divider value (1-8)
<i>post</i>	Post divider value (1-64)

### 5.5.8 static uint32\_t CLOCK\_GetRootPreDivider ( clock\_root\_control\_t *rootClk* ) [inline], [static]

In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

<i>rootClk</i>	Root clock name (see <a href="#">clock_root_control_t</a> enumeration).
----------------	---

Returns

Root Pre divider value.

### 5.5.9 static uint32\_t CLOCK\_GetRootPostDivider ( clock\_root\_control\_t *rootClk* ) [inline], [static]

In order to get the clock source of root, user maybe need to get more than one ROOT's mux value to obtain the final clock source of root.

Parameters

<i>rootClk</i>	Root clock name (see <a href="#">clock_root_control_t</a> enumeration).
----------------	---

Returns

Root Post divider value.

#### 5.5.10 void CLOCK\_InitOSC25M ( const osc\_config\_t \* *config* )

Parameters

<i>config</i>	osc configuration
---------------	-------------------

#### 5.5.11 void CLOCK\_DeinitOSC25M ( void )

#### 5.5.12 void CLOCK\_InitOSC27M ( const osc\_config\_t \* *config* )

#### 5.5.13 void CLOCK\_DeinitOSC27M ( void )

Parameters

<i>config</i>	osc configuration
---------------	-------------------

#### 5.5.14 static void CLOCK\_SwitchOSC32Src ( osc32\_src\_t *sel* ) [inline], [static]

Parameters

<i>sel</i>	OSC32 input clock select
------------	--------------------------

#### 5.5.15 static void CLOCK\_ControlGate ( uint32\_t *ccmGate*, clock\_gate\_value\_t *control* ) [inline], [static]

base CCM base pointer.

## Function Documentation

### Parameters

<i>ccmGate</i>	Gate control (see <a href="#">clock_pll_gate_t</a> and <a href="#">clock_ip_name_t</a> enumeration)
<i>control</i>	Gate control value (see <a href="#">clock_gate_value_t</a> )

### 5.5.16 void CLOCK\_EnableClock ( clock\_ip\_name\_t *ccmGate* )

Take care of that one module may need to set more than one clock gate.

### Parameters

<i>ccmGate</i>	Gate control for each module (see <a href="#">clock_ip_name_t</a> enumeration).
----------------	---

### 5.5.17 void CLOCK\_DisableClock ( clock\_ip\_name\_t *ccmGate* )

Take care of that one module may need to set more than one clock gate.

### Parameters

<i>ccmGate</i>	Gate control for each module (see <a href="#">clock_ip_name_t</a> enumeration).
----------------	---

### 5.5.18 static void CLOCK\_PowerUpPll ( CCM\_ANALOG\_Type \* *base*, clock\_pll\_ctrl\_t *pllControl* ) [inline], [static]

### Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>pllControl</i>	PLL control name (see <a href="#">clock_pll_ctrl_t</a> enumeration)

### 5.5.19 static void CLOCK\_PowerDownPll ( CCM\_ANALOG\_Type \* *base*, clock\_pll\_ctrl\_t *pllControl* ) [inline], [static]

### Parameters

---

<i>base</i>	CCM_ANALOG base pointer.
<i>pllControl</i>	PLL control name (see <a href="#">clock_pll_ctrl_t</a> enumeration)

**5.5.20 static void CLOCK\_SetPIIBypass ( CCM\_ANALOG\_Type \* *base*,  
clock\_pll\_bypass\_ctrl\_t *pllControl*, bool *bypass* ) [inline], [static]**

Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>pllControl</i>	PLL control name (see <a href="#">ccm_analog_pll_control_t</a> enumeration)
<i>bypass</i>	Bypass the PLL. <ul style="list-style-type: none"> <li>• true: Bypass the PLL.</li> <li>• false: Do not bypass the PLL.</li> </ul>

**5.5.21 static bool CLOCK\_IsPIIBypassed ( CCM\_ANALOG\_Type \* *base*,  
clock\_pll\_bypass\_ctrl\_t *pllControl* ) [inline], [static]**

Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>pllControl</i>	PLL control name (see <a href="#">ccm_analog_pll_control_t</a> enumeration)

Returns

PLL bypass status.

- true: The PLL is bypassed.
- false: The PLL is not bypassed.

**5.5.22 static bool CLOCK\_IsPIILocked ( CCM\_ANALOG\_Type \* *base*,  
clock\_pll\_ctrl\_t *pllControl* ) [inline], [static]**

## Function Documentation

### Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>pllControl</i>	PLL control name (see <a href="#">clock_pll_ctrl_t</a> enumeration)

### Returns

PLL lock status.

- true: The PLL clock is locked.
- false: The PLL clock is not locked.

**5.5.23 static void CLOCK\_EnableAnalogClock ( CCM\_ANALOG\_Type \* *base*,  
clock\_pll\_clke\_t *pllClock* ) [inline], [static]**

### Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>pllClock</i>	PLL clock name (see <a href="#">ccm_analog_pll_clock_t</a> enumeration)

**5.5.24 static void CLOCK\_DisableAnalogClock ( CCM\_ANALOG\_Type \* *base*,  
clock\_pll\_clke\_t *pllClock* ) [inline], [static]**

### Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>pllClock</i>	PLL clock name (see <a href="#">ccm_analog_pll_clock_t</a> enumeration)

**5.5.25 static void CLOCK\_OverrideAnalogClke ( CCM\_ANALOG\_Type \* *base*,  
clock\_pll\_clke\_t *ovClock*, bool *override* ) [inline], [static]**

### Parameters

---

<i>base</i>	CCM_ANALOG base pointer.
<i>ovClock</i>	PLL clock name (see <a href="#">clock_pll_clke_t</a> enumeration)
<i>override</i>	Override the PLL. <ul style="list-style-type: none"> <li>• true: Override the PLL clke, CCM will handle it.</li> <li>• false: Do not override the PLL clke.</li> </ul>

#### 5.5.26 static void CLOCK\_OverridePIIPd ( CCM\_ANALOG\_Type \* *base*, clock\_pll\_ctrl\_t *pdClock*, bool *override* ) [inline], [static]

Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>pdClock</i>	PLL clock name (see <a href="#">clock_pll_ctrl_t</a> enumeration)
<i>override</i>	Override the PLL. <ul style="list-style-type: none"> <li>• true: Override the PLL clke, CCM will handle it.</li> <li>• false: Do not override the PLL clke.</li> </ul>

#### 5.5.27 void CLOCK\_InitArmPll ( const ccm\_analog\_frac\_pll\_config\_t \* *config* )

Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_frac_pll_config_t</a> enumeration).
---------------	---

Note

This function can't detect whether the Arm PLL has been enabled and used by some IPs.

#### 5.5.28 void CLOCK\_InitSysPll1 ( const ccm\_analog\_sscg\_pll\_config\_t \* *config* )

## Function Documentation

### Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_sscg_pll_config_t</a> enumeration).
---------------	---

### Note

This function can't detect whether the SYS PLL has been enabled and used by some IPs.

### 5.5.29 void CLOCK\_InitSysPII2 ( const ccm\_analog\_sscg\_pll\_config\_t \* *config* )

### Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_sscg_pll_config_t</a> enumeration).
---------------	---

### Note

This function can't detect whether the SYS PLL has been enabled and used by some IPs.

### 5.5.30 void CLOCK\_InitSysPII3 ( const ccm\_analog\_sscg\_pll\_config\_t \* *config* )

### Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_sscg_pll_config_t</a> enumeration).
---------------	---

### Note

This function can't detect whether the SYS PLL has been enabled and used by some IPs.

### 5.5.31 void CLOCK\_InitDramPII ( const ccm\_analog\_sscg\_pll\_config\_t \* *config* )



## Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_sscg_pll_config_t</a> enumeration).
---------------	---

## Note

This function can't detect whether the DDR PLL has been enabled and used by some IPs.

### 5.5.32 void CLOCK\_InitAudioPII1 ( const ccm\_analog\_frac\_pll\_config\_t \* *config* )

## Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_frac_pll_config_t</a> enumeration).
---------------	---

## Note

This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

### 5.5.33 void CLOCK\_InitAudioPII2 ( const ccm\_analog\_frac\_pll\_config\_t \* *config* )

## Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_frac_pll_config_t</a> enumeration).
---------------	---

## Note

This function can't detect whether the AUDIO PLL has been enabled and used by some IPs.

### 5.5.34 void CLOCK\_InitVideoPII1 ( const ccm\_analog\_frac\_pll\_config\_t \* *config* )

## Function Documentation

### Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_frac_pll_config_t</a> enumeration).
---------------	---

### 5.5.35 void CLOCK\_InitVideoPll2 ( const ccm\_analog\_sscg\_pll\_config\_t \* *config* )

### Parameters

<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_sscg_pll_config_t</a> enumeration).
---------------	---

### Note

This function can't detect whether the VIDEO PLL has been enabled and used by some IPs.

### 5.5.36 void CLOCK\_InitSSCGPll ( CCM\_ANALOG\_Type \* *base*, const ccm\_analog\_sscg\_pll\_config\_t \* *config*, clock\_pll\_ctrl\_t *type* )

### Parameters

<i>base</i>	CCM ANALOG base address
<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_sscg_pll_config_t</a> enumeration).
<i>type</i>	sscg pll type

### 5.5.37 uint32\_t CLOCK\_GetSSCGPllFreq ( CCM\_ANALOG\_Type \* *base*, clock\_pll\_ctrl\_t *type*, uint32\_t *refClkFreq*, bool *pll1Bypass* )

### Parameters

<i>base</i>	CCM ANALOG base address.
-------------	--------------------------

<i>type</i>	sscg pll type
<i>pll1Bypass</i>	pll1 bypass flag

Returns

Clock frequency

**5.5.38 void CLOCK\_InitFracPll ( CCM\_ANALOG\_Type \* *base*, const ccm\_analog\_frac\_pll\_config\_t \* *config*, clock\_pll\_ctrl\_t *type* )**

Parameters

<i>base</i>	CCM ANALOG base address.
<i>config</i>	Pointer to the configuration structure(see <a href="#">ccm_analog_frac_pll_config_t</a> enumeration).
<i>type</i>	fractional pll type.

**5.5.39 uint32\_t CLOCK\_GetFracPllFreq ( CCM\_ANALOG\_Type \* *base*, clock\_pll\_ctrl\_t *type*, uint32\_t *refClkFreq* )**

Parameters

<i>base</i>	CCM_ANALOG base pointer.
<i>type</i>	fractional pll type.
<i>fractional</i>	pll reference clock frequency

Returns

Clock frequency

**5.5.40 uint32\_t CLOCK\_GetPllFreq ( clock\_pll\_ctrl\_t *pll* )**

## Function Documentation

### Parameters

<i>type</i>	fractional pll type.
-------------	----------------------

### Returns

Clock frequency

#### 5.5.41 uint32\_t CLOCK\_GetPllRefClkFreq ( clock\_pll\_ctrl\_t *ctrl* )

### Parameters

<i>type</i>	fractional pll type.
-------------	----------------------

### Returns

Clock frequency

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

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock\_name\_t.

### Parameters

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

### Returns

Clock frequency value in hertz

#### 5.5.43 uint32\_t CLOCK\_GetCoreM4Freq ( void )

### Returns

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

#### 5.5.44 uint32\_t CLOCK\_GetAxiFreq ( void )

Returns

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

#### 5.5.45 uint32\_t CLOCK\_GetAhbFreq ( void )

Returns

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





## Chapter 6

# ECSPI: Serial Peripheral Interface Driver

### 6.1 Overview

#### Modules

- [ECSPI Driver](#)
- [ECSPI FreeRTOS Driver](#)

## 6.2 ECSPI Driver

### 6.2.1 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.

### 6.2.2 Typical use case

#### 6.2.2.1 SPI master transfer using polling method

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

#### 6.2.2.2 SPI master transfer using an interrupt method

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

## 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_TxfifoEmptyFlag = ECSPI_STATREG_TE_MASK,
kECSPI_TxFifoDataRequestFlag = ECSPI_STATREG_TDR_MASK,
kECSPI_TxFifoFullFlag = ECSPI_STATREG_TF_MASK,
kECSPI_RxFifoReadyFlag = ECSPI_STATREG_RR_MASK,
kECSPI_RxFifoDataRequestFlag = ECSPI_STATREG_RDR_MASK,
kECSPI_RxFifoFullFlag = ECSPI_STATREG_RF_MASK,
kECSPI_RxFifoOverflowFlag = 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.*

- 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.*

## Functions

- `uint32_t ECSPI_GetInstance (ECSPI_Type *base)`  
*Get the instance for ECSPI module.*

## 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.*

## ECSPI Driver

### 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.*
- static void [ECSPI\\_WriteData](#) (ECSPI\_Type \*base, uint32\_t data)  
*Writes a data into the ECSPI data register.*
- static uint32\_t [ECSPI\\_ReadData](#) (ECSPI\_Type \*base)  
*Gets a data from the ECSPI data register.*

### Transactional

- void [ECSPI\\_MasterTransferCreateHandle](#) (ECSPI\_Type \*base, [ecspi\\_master\\_handle\\_t](#) \*handle, [ecspi\\_master\\_callback\\_t](#) callback, void \*userData)  
*Initializes the ECSPI master handle.*
- status\_t [ECSPI\\_MasterTransferBlocking](#) (ECSPI\_Type \*base, [ecspi\\_transfer\\_t](#) \*xfer)

- Transfers a block of data using a polling method.
- `status_t ECSPI_MasterTransferNonBlocking` (`ECSPI_Type *base`, `ecspi_master_handle_t *handle`, `ecspi_transfer_t *xfer`)
- Performs a non-blocking ECSPI interrupt transfer.
- `status_t ECSPI_MasterTransferGetCount` (`ECSPI_Type *base`, `ecspi_master_handle_t *handle`, `size_t *count`)
- Gets the bytes of the ECSPI interrupt transferred.
- `void ECSPI_MasterTransferAbort` (`ECSPI_Type *base`, `ecspi_master_handle_t *handle`)
- Aborts an ECSPI transfer using interrupt.
- `void ECSPI_MasterTransferHandleIRQ` (`ECSPI_Type *base`, `ecspi_master_handle_t *handle`)
- Interrupts the handler for the ECSPI.
- `void ECSPI_SlaveTransferCreateHandle` (`ECSPI_Type *base`, `ecspi_slave_handle_t *handle`, `ecspi_slave_callback_t callback`, `void *userData`)
- Initializes the ECSPI slave handle.
- `static status_t ECSPI_SlaveTransferNonBlocking` (`ECSPI_Type *base`, `ecspi_slave_handle_t *handle`, `ecspi_transfer_t *xfer`)
- Performs a non-blocking ECSPI slave interrupt transfer.
- `static status_t ECSPI_SlaveTransferGetCount` (`ECSPI_Type *base`, `ecspi_slave_handle_t *handle`, `size_t *count`)
- Gets the bytes of the ECSPI interrupt transferred.
- `static void ECSPI_SlaveTransferAbort` (`ECSPI_Type *base`, `ecspi_slave_handle_t *handle`)
- Aborts an ECSPI slave transfer using interrupt.
- `void ECSPI_SlaveTransferHandleIRQ` (`ECSPI_Type *base`, `ecspi_slave_handle_t *handle`)
- Interrupts a handler for the ECSPI slave.

## 6.2.3 Data Structure Documentation

### 6.2.3.1 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 chipSlectActiveState`  
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.

### 6.2.3.2 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.*
- [bool enableLoopback](#)  
*Enable the ECSPI loopback test.*

#### 6.2.3.2.0.1 Field Documentation

##### 6.2.3.2.0.1.1 bool ecspi\_master\_config\_t::enableLoopback

### 6.2.3.3 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.*

### 6.2.3.4 struct ecspi\_transfer\_t

#### Data Fields

- [uint32\\_t \\* txData](#)  
*Send buffer.*
- [uint32\\_t \\* rxData](#)  
*Receive buffer.*

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

### 6.2.3.5 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.*

### 6.2.4 Macro Definition Documentation

6.2.4.1 `#define FSL_ECSPi_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

6.2.4.2 `#define ECSPi_DUMMYDATA (0xFFFFFFFFFU)`

### 6.2.5 Enumeration Type Documentation

#### 6.2.5.1 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.

## ECSPI Driver

### 6.2.5.2 enum ecspi\_clock\_polarity\_t

Enumerator

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

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

### 6.2.5.3 enum ecspi\_clock\_phase\_t

Enumerator

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

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

### 6.2.5.4 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.

### 6.2.5.5 enum \_ecspi\_flags

Enumerator

***kECSPI\_TxfifoEmptyFlag*** 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.



### 6.2.5.6 enum\_ecspi\_dma\_enable\_t

Enumerator

*kECSPI\_TxDmaEnable* Tx DMA request source.  
*kECSPI\_RxDmaEnable* Rx DMA request source.  
*kECSPI\_DmaAllEnable* All DMA request source.

### 6.2.5.7 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.

### 6.2.5.8 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.

### 6.2.5.9 enum\_ecspi\_master\_slave\_mode\_t

Enumerator

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

### 6.2.5.10 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.

## ECSPI Driver

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

Enumerator

*kECSPI\_ClockInactiveStateLow* The SCLK inactive state stays low.

*kECSPI\_ClockInactiveStateHigh* The SCLK inactive state stays high.

### 6.2.5.12 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.

### 6.2.5.13 enum ecspi\_wave\_form\_t

Enumerator

*kECSPI\_WaveFormSingle* The wave form for signal burst.

*kECSPI\_WaveFormMultiple* The wave form for multiple burst.

### 6.2.5.14 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).

## 6.2.6 Function Documentation

### 6.2.6.1 uint32\_t ECSPI\_GetInstance ( ECSPI\_Type \* *base* )

Parameters

<i>base</i>	ECSPI base address
-------------	--------------------

### 6.2.6.2 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);
```

Parameters

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

### 6.2.6.3 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.

### 6.2.6.4 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
---------------	-----------------------------

### 6.2.6.5 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 Driver

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

### Parameters

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

### 6.2.6.6 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
-------------	--------------------

### 6.2.6.7 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

### 6.2.6.8 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.

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

Parameters

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

**6.2.6.10 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_TxfifoEmptyInterruptEnable</li> <li>• kECSPI_TxFifoDataRequestInterruptEnable</li> <li>• kECSPI_TxFifoFullInterruptEnable</li> <li>• kECSPI_RxFifoReadyInterruptEnable</li> <li>• kECSPI_RxFifoDataRequestInterruptEnable</li> <li>• kECSPI_RxFifoFullInterruptEnable</li> <li>• kECSPI_RxFifoOverflowInterruptEnable</li> <li>• kECSPI_TransferCompleteInterruptEnable</li> <li>• kECSPI_AllInterruptEnable</li> </ul>

## ECSPI Driver

**6.2.6.11 static void ECSPI\_DisableInterrupts ( 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_TxfifoEmptyInterruptEnable</li><li>• kECSPI_TxFifoDataRequestInterruptEnable</li><li>• kECSPI_TxFifoFullInterruptEnable</li><li>• kECSPI_RxFifoReadyInterruptEnable</li><li>• kECSPI_RxFifoDataRequestInterruptEnable</li><li>• kECSPI_RxFifoFullInterruptEnable</li><li>• kECSPI_RxFifoOverflowInterruptEnable</li><li>• kECSPI_TransferCompleteInterruptEnable</li><li>• kECSPI_AllInterruptEnable</li></ul>

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

Parameters

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

**6.2.6.13 static bool ECSPI\_IsMaster ( ECSPI\_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

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

Parameters

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

### 6.2.6.15 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.

### 6.2.6.16 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.

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

Parameters

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

## ECSPI Driver

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

### 6.2.6.18 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

### 6.2.6.19 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.

### 6.2.6.20 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
-------------	--------------------



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

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

Parameters

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

**6.2.6.22 static uint32\_t ECSPI\_ReadData ( ECSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

Data in the register.

**6.2.6.23 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.

## ECSPI Driver

<i>userData</i>	User data.
-----------------	------------

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

#### Parameters

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

#### Return values

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

### 6.2.6.25 **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 ecspi_master_handle_t structure which stores the transfer state
<i>xfer</i>	pointer to <a href="#">ecspi_transfer_t</a> 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.

### 6.2.6.26 **status\_t** ECSPI\_MasterTransferGetCount ( **ECSPI\_Type** \* *base*, **ecspi\_master\_handle\_t** \* *handle*, **size\_t** \* *count* )

#### Parameters

<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.

#### 6.2.6.27 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.

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

#### Parameters

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

#### 6.2.6.29 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.

## ECSPI Driver

### Parameters

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

**6.2.6.30** `static status_t ECSPI_SlaveTransferNonBlocking ( 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 <code>ecspi_master_handle_t</code> structure which stores the transfer state
<i>xfer</i>	pointer to <a href="#">ecspi_transfer_t</a> 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.

**6.2.6.31** `static status_t ECSPI_SlaveTransferGetCount ( ECSPI_Type * base,  
ecspi_slave_handle_t * handle, size_t * count ) [inline], [static]`

### Parameters

<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 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.

**6.2.6.32** `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.

**6.2.6.33** `void ECSPI_SlaveTransferHandleIRQ ( ECSPI_Type * base, ecspi_slave_handle_t  
* handle )`

Parameters

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

### 6.3 ECSPI FreeRTOS Driver

#### 6.3.1 Overview

##### Driver version

- #define **FSL\_ECSPI\_FREERTOS\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 0))  
*ECSPI freertos driver version 2.0.0.*

#### 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.*

#### 6.3.2 Macro Definition Documentation

##### 6.3.2.1 #define FSL\_ECSPI\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

#### 6.3.3 Function Documentation

##### 6.3.3.1 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.

### 6.3.3.2 `status_t ECSPI_RTOS_Deinit ( ecspi_rtos_handle_t * handle )`

This function deinitializes the ECSPI module and related RTOS context.

## ECSPI FreeRTOS Driver

### Parameters

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

### 6.3.3.3 **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 7

# GPT: General Purpose Timer

### 7.1 Overview

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

### 7.2 Function groups

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

#### 7.2.1 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.

### 7.3 Typical use case

#### 7.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/gpt`

### Data Structures

- struct [gpt\\_config\\_t](#)  
*Structure to configure the running mode. [More...](#)*

### Enumerations

- enum [gpt\\_clock\\_source\\_t](#) {  
    [kGPT\\_ClockSource\\_Off](#) = 0U,  
    [kGPT\\_ClockSource\\_Periph](#) = 1U,  
    [kGPT\\_ClockSource\\_HighFreq](#) = 2U,  
    [kGPT\\_ClockSource\\_Ext](#) = 3U,  
    [kGPT\\_ClockSource\\_LowFreq](#) = 4U,  
    [kGPT\\_ClockSource\\_Osc](#) = 5U }  
*List of clock sources.*

## Typical use case

- enum `gpt_input_capture_channel_t` {  
    `kGPT_InputCapture_Channel1` = 0U,  
    `kGPT_InputCapture_Channel2` = 1U }  
    *List of input capture channel number.*
- enum `gpt_input_operation_mode_t` {  
    `kGPT_InputOperation_Disabled` = 0U,  
    `kGPT_InputOperation_RiseEdge` = 1U,  
    `kGPT_InputOperation_FallEdge` = 2U,  
    `kGPT_InputOperation_BothEdge` = 3U }  
    *List of input capture operation mode.*
- enum `gpt_output_compare_channel_t` {  
    `kGPT_OutputCompare_Channel1` = 0U,  
    `kGPT_OutputCompare_Channel2` = 1U,  
    `kGPT_OutputCompare_Channel3` = 2U }  
    *List of output compare channel number.*
- enum `gpt_output_operation_mode_t` {  
    `kGPT_OutputOperation_Disconnected` = 0U,  
    `kGPT_OutputOperation_Toggle` = 1U,  
    `kGPT_OutputOperation_Clear` = 2U,  
    `kGPT_OutputOperation_Set` = 3U,  
    `kGPT_OutputOperation_Activelow` = 4U }  
    *List of output compare operation mode.*
- enum `gpt_interrupt_enable_t` {  
    `kGPT_OutputCompare1InterruptEnable` = GPT\_IR\_OF1IE\_MASK,  
    `kGPT_OutputCompare2InterruptEnable` = GPT\_IR\_OF2IE\_MASK,  
    `kGPT_OutputCompare3InterruptEnable` = GPT\_IR\_OF3IE\_MASK,  
    `kGPT_InputCapture1InterruptEnable` = GPT\_IR\_IF1IE\_MASK,  
    `kGPT_InputCapture2InterruptEnable` = GPT\_IR\_IF2IE\_MASK,  
    `kGPT_RollOverFlagInterruptEnable` = GPT\_IR\_ROVIE\_MASK }  
    *List of GPT interrupts.*
- enum `gpt_status_flag_t` {  
    `kGPT_OutputCompare1Flag` = GPT\_SR\_OF1\_MASK,  
    `kGPT_OutputCompare2Flag` = GPT\_SR\_OF2\_MASK,  
    `kGPT_OutputCompare3Flag` = GPT\_SR\_OF3\_MASK,  
    `kGPT_InputCapture1Flag` = GPT\_SR\_IF1\_MASK,  
    `kGPT_InputCapture2Flag` = GPT\_SR\_IF2\_MASK,  
    `kGPT_RollOverFlag` = GPT\_SR\_ROV\_MASK }  
    *Status flag.*

## Driver version

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

## Initialization and deinitialization

- void `GPT_Init` (GPT\_Type \*base, const `gpt_config_t` \*initConfig)

- *Initialize GPT to reset state and initialize running mode.*
- void [GPT\\_Deinit](#) (GPT\_Type \*base)  
*Disables the module and gates the GPT clock.*
- void [GPT\\_GetDefaultConfig](#) (gpt\_config\_t \*config)  
*Fills in the GPT configuration structure with default settings.*

## Software Reset

- static void [GPT\\_SoftwareReset](#) (GPT\_Type \*base)  
*Software reset of GPT module.*

## Clock source and frequency control

- static void [GPT\\_SetClockSource](#) (GPT\_Type \*base, gpt\_clock\_source\_t source)  
*Set clock source of GPT.*
- static gpt\_clock\_source\_t [GPT\\_GetClockSource](#) (GPT\_Type \*base)  
*Get clock source of GPT.*
- static void [GPT\\_SetClockDivider](#) (GPT\_Type \*base, uint32\_t divider)  
*Set pre scaler of GPT.*
- static uint32\_t [GPT\\_GetClockDivider](#) (GPT\_Type \*base)  
*Get clock divider in GPT module.*
- static void [GPT\\_SetOscClockDivider](#) (GPT\_Type \*base, uint32\_t divider)  
*OSC 24M pre-scaler before selected by clock source.*
- static uint32\_t [GPT\\_GetOscClockDivider](#) (GPT\_Type \*base)  
*Get OSC 24M clock divider in GPT module.*

## Timer Start and Stop

- static void [GPT\\_StartTimer](#) (GPT\_Type \*base)  
*Start GPT timer.*
- static void [GPT\\_StopTimer](#) (GPT\_Type \*base)  
*Stop GPT timer.*

## Read the timer period

- static uint32\_t [GPT\\_GetCurrentTimerCount](#) (GPT\_Type \*base)  
*Reads the current GPT counting value.*

## GPT Input/Output Signal Control

- static void [GPT\\_SetInputOperationMode](#) (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel, gpt\_input\_operation\_mode\_t mode)  
*Set GPT operation mode of input capture channel.*
- static gpt\_input\_operation\_mode\_t [GPT\\_GetInputOperationMode](#) (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel)  
*Get GPT operation mode of input capture channel.*
- static uint32\_t [GPT\\_GetInputCaptureValue](#) (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel)  
*Get GPT input capture value of certain channel.*

## Data Structure Documentation

- static void [GPT\\_SetOutputOperationMode](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel, [gpt\\_output\\_operation\\_mode\\_t](#) mode)  
*Set GPT operation mode of output compare channel.*
- static [gpt\\_output\\_operation\\_mode\\_t](#) [GPT\\_GetOutputOperationMode](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)  
*Get GPT operation mode of output compare channel.*
- static void [GPT\\_SetOutputCompareValue](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel, uint32\_t value)  
*Set GPT output compare value of output compare channel.*
- static uint32\_t [GPT\\_GetOutputCompareValue](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)  
*Get GPT output compare value of output compare channel.*
- static void [GPT\\_ForceOutput](#) (GPT\_Type \*base, [gpt\\_output\\_compare\\_channel\\_t](#) channel)  
*Force GPT output action on output compare channel, ignoring comparator.*

## GPT Interrupt and Status Interface

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

## Status Interface

- static uint32\_t [GPT\\_GetStatusFlags](#) (GPT\_Type \*base, [gpt\\_status\\_flag\\_t](#) flags)  
*Get GPT status flags.*
- static void [GPT\\_ClearStatusFlags](#) (GPT\_Type \*base, [gpt\\_status\\_flag\\_t](#) flags)  
*Clears the GPT status flags.*

## 7.4 Data Structure Documentation

### 7.4.1 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.

#### 7.4.1.0.0.2 Field Documentation

7.4.1.0.0.2.1 `gpt_clock_source_t gpt_config_t::clockSource`

7.4.1.0.0.2.2 `uint32_t gpt_config_t::divider`

7.4.1.0.0.2.3 `bool gpt_config_t::enableFreeRun`

7.4.1.0.0.2.4 `bool gpt_config_t::enableRunInWait`

7.4.1.0.0.2.5 `bool gpt_config_t::enableRunInStop`

7.4.1.0.0.2.6 `bool gpt_config_t::enableRunInDoze`

7.4.1.0.0.2.7 `bool gpt_config_t::enableRunInDbg`

7.4.1.0.0.2.8 `bool gpt_config_t::enableMode`

### 7.5 Enumeration Type Documentation

#### 7.5.1 `enum gpt_clock_source_t`

Note

Actual number of clock sources is SoC dependent

Enumerator

***kGPT\_ClockSource\_Off*** GPT Clock Source Off.  
***kGPT\_ClockSource\_Periph*** GPT Clock Source from Peripheral Clock.  
***kGPT\_ClockSource\_HighFreq*** GPT Clock Source from High Frequency Reference Clock.  
***kGPT\_ClockSource\_Ext*** GPT Clock Source from external pin.  
***kGPT\_ClockSource\_LowFreq*** GPT Clock Source from Low Frequency Reference Clock.  
***kGPT\_ClockSource\_Osc*** GPT Clock Source from Crystal oscillator.

#### 7.5.2 `enum gpt_input_capture_channel_t`

Enumerator

***kGPT\_InputCapture\_Channel1*** GPT Input Capture Channel1.  
***kGPT\_InputCapture\_Channel2*** GPT Input Capture Channel2.

### 7.5.3 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.

### 7.5.4 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.

### 7.5.5 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.

### 7.5.6 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.

## 7.5.7 enum gpt\_status\_flag\_t

Enumerator

***kGPT\_OutputCompare1Flag*** Output compare channel 1 event.  
***kGPT\_OutputCompare2Flag*** Output compare channel 2 event.  
***kGPT\_OutputCompare3Flag*** Output compare channel 3 event.  
***kGPT\_InputCapture1Flag*** Input Capture channel 1 event.  
***kGPT\_InputCapture2Flag*** Input Capture channel 2 event.  
***kGPT\_RollOverFlag*** Counter reaches maximum value and rolled over to 0 event.

## 7.6 Function Documentation

### 7.6.1 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.

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

Parameters

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

### 7.6.3 void GPT\_GetDefaultConfig ( gpt\_config\_t \* *config* )

The default values are:

```

*  config->clockSource = kGPT_ClockSource_Periph;
*  config->divider = 1U;
*  config->enableRunInStop = true;
*  config->enableRunInWait = true;
*  config->enableRunInDoze = false;
*  config->enableRunInDbg = false;
*  config->enableFreeRun = true;
*  config->enableMode = true;
*

```

## Function Documentation

### Parameters

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

### 7.6.4 static void GPT\_SoftwareReset ( GPT\_Type \* *base* ) [inline], [static]

### Parameters

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

### 7.6.5 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).

### 7.6.6 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).

### 7.6.7 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).

### 7.6.8 static uint32\_t GPT\_GetClockDivider ( GPT\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

clock divider in GPT module (1-4096).

### 7.6.9 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).

### 7.6.10 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).

### 7.6.11 static void GPT\_StartTimer ( GPT\_Type \* *base* ) [inline], [static]

## Function Documentation

### Parameters

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

### 7.6.12 static void GPT\_StopTimer ( GPT\_Type \* *base* ) [inline], [static]

### Parameters

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

### 7.6.13 static uint32\_t GPT\_GetCurrentTimerCount ( GPT\_Type \* *base* ) [inline], [static]

### Parameters

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

### Returns

Current GPT counter value.

### 7.6.14 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).

### 7.6.15 static gpt\_input\_operation\_mode\_t GPT\_GetInputOperationMode ( GPT\_Type \* *base*, gpt\_input\_capture\_channel\_t *channel* ) [inline], [static]

## Parameters

<i>base</i>	GPT peripheral base address.
<i>channel</i>	GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration).

## Returns

GPT input capture operation mode (see [gpt\\_input\\_operation\\_mode\\_t](#) typedef enumeration).

### 7.6.16 **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.

### 7.6.17 **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).

### 7.6.18 **static gpt\_output\_operation\_mode\_t GPT\_GetOutputOperationMode ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel* ) [inline], [static]**

## Function Documentation

### Parameters

<i>base</i>	GPT peripheral base address.
<i>channel</i>	GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration).

### Returns

GPT output operation mode (see [gpt\\_output\\_operation\\_mode\\_t](#) typedef enumeration).

**7.6.19 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.

**7.6.20 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.

**7.6.21 static void GPT\_ForceOutput ( GPT\_Type \* *base*, gpt\_output\_compare\_-  
channel\_t *channel* ) [inline], [static]**

## Parameters

<i>base</i>	GPT peripheral base address.
<i>channel</i>	GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration).

### 7.6.22 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>

### 7.6.23 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>

### 7.6.24 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](#)

**7.6.25** `static uint32_t GPT_GetStatusFlags ( GPT_Type * base, gpt_status_flag_t flags )` `[inline]`, `[static]`

## Parameters

<i>base</i>	GPT peripheral base address.
<i>flags</i>	GPT status flag mask (see <a href="#">gpt_status_flag_t</a> for bit definition).

## Returns

GPT status, each bit represents one status flag.

### 7.6.26 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 8

# GPIO: General-Purpose Input/Output Driver

### 8.1 Overview

#### Modules

- [GPIO Driver](#)

## 8.2 GPIO Driver

### 8.2.1 Overview

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

### 8.2.2 Typical use case

#### 8.2.2.1 Input Operation

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

## Data Structures

- struct [gpio\\_pin\\_config\\_t](#)  
*GPIO Init structure definition. [More...](#)*

## Enumerations

- enum [gpio\\_pin\\_direction\\_t](#) {  
    [kGPIO\\_DigitalInput](#) = 0U,  
    [kGPIO\\_DigitalOutput](#) = 1U }  
*GPIO direction definition.*
- enum [gpio\\_interrupt\\_mode\\_t](#) {  
    [kGPIO\\_NoIntmode](#) = 0U,  
    [kGPIO\\_IntLowLevel](#) = 1U,  
    [kGPIO\\_IntHighLevel](#) = 2U,  
    [kGPIO\\_IntRisingEdge](#) = 3U,  
    [kGPIO\\_IntFallingEdge](#) = 4U,  
    [kGPIO\\_IntRisingOrFallingEdge](#) = 5U }  
*GPIO interrupt mode definition.*

## Driver version

- #define [FSL\\_GPIO\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 1))  
*GPIO driver version 2.0.1.*

## GPIO Initialization and Configuration functions

- void [GPIO\\_PinInit](#) (GPIO\_Type \*base, uint32\_t pin, const [gpio\\_pin\\_config\\_t](#) \*Config)  
*Initializes the GPIO peripheral according to the specified parameters in the initConfig.*

## GPIO Reads and Write Functions

- void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void [GPIO\\_WritePinOutput](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_SetPinsOutput](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_PortClear](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_ClearPinsOutput](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_PortToggle](#) (GPIO\_Type \*base, uint32\_t mask)  
*Reverses the current output logic of the multiple GPIO pins.*
- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*
- static uint32\_t [GPIO\\_ReadPinInput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*

## GPIO Reads Pad Status Functions

- static uint8\_t [GPIO\\_PinReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*
- static uint8\_t [GPIO\\_ReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*

## Interrupts and flags management functions

- void [GPIO\\_PinSetInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_SetPinInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_PortEnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_EnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_PortDisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static void [GPIO\\_DisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static uint32\_t [GPIO\\_PortGetInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*
- static uint32\_t [GPIO\\_GetPinsInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*

## GPIO Driver

- static void [GPIO\\_PortClearInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*
- static void [GPIO\\_ClearPinsInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*

### 8.2.3 Data Structure Documentation

#### 8.2.3.1 struct gpio\_pin\_config\_t

##### Data Fields

- [gpio\\_pin\\_direction\\_t direction](#)  
*Specifies the pin direction.*
- uint8\_t [outputLogic](#)  
*Set a default output logic, which has no use in input.*
- [gpio\\_interrupt\\_mode\\_t interruptMode](#)  
*Specifies the pin interrupt mode, a value of [gpio\\_interrupt\\_mode\\_t](#).*

##### 8.2.3.1.0.3 Field Documentation

###### 8.2.3.1.0.3.1 gpio\_pin\_direction\_t gpio\_pin\_config\_t::direction

###### 8.2.3.1.0.3.2 gpio\_interrupt\_mode\_t gpio\_pin\_config\_t::interruptMode

### 8.2.4 Macro Definition Documentation

#### 8.2.4.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 8.2.5 Enumeration Type Documentation

#### 8.2.5.1 enum gpio\_pin\_direction\_t

Enumerator

***kGPIO\_DigitalInput*** Set current pin as digital input.  
***kGPIO\_DigitalOutput*** Set current pin as digital output.

#### 8.2.5.2 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.

## 8.2.6 Function Documentation

### 8.2.6.1 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.

### 8.2.6.2 void GPIO\_PinWrite ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* )

Parameters

<i>base</i>	GPIO base pointer.
<i>pin</i>	GPIO port pin number.
<i>output</i>	GPIOpin output logic level. <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul>

### 8.2.6.3 static void GPIO\_WritePinOutput ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

### 8.2.6.4 static void GPIO\_PortSet ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

## GPIO Driver

<i>base</i>	GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
<i>mask</i>	GPIO pin number macro

**8.2.6.5 static void GPIO\_SetPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

**8.2.6.6 static void GPIO\_PortClear ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline],**  
**[static]**

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
<i>mask</i>	GPIO pin number macro

**8.2.6.7 static void GPIO\_ClearPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

**8.2.6.8 static void GPIO\_PortToggle ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline],**  
**[static]**

Parameters

<i>base</i>	GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.)
<i>mask</i>	GPIO pin number macro

**8.2.6.9 static uint32\_t GPIO\_PinRead ( GPIO\_Type \* *base*, uint32\_t *pin* ) [inline],**  
**[static]**

Parameters

<i>base</i>	GPIO base pointer.
<i>pin</i>	GPIO port pin number.

Return values

---

<i>GPIO</i>	port input value.
-------------	-------------------

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

**8.2.6.11** `static uint8_t GPIO_PinReadPadStatus ( GPIO_Type * base, uint32_t pin )`  
`[inline], [static]`

Parameters

<i>base</i>	GPIO base pointer.
<i>pin</i>	GPIO port pin number.

Return values

<i>GPIO</i>	pin pad status value.
-------------	-----------------------

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

**8.2.6.13** `void GPIO_PinSetInterruptConfig ( GPIO_Type * base, uint32_t pin,  
gpio_interrupt_mode_t pinInterruptMode )`

Parameters

<i>base</i>	GPIO base pointer.
<i>pin</i>	GPIO port pin number.
<i>pininterrupt-Mode</i>	pointer to a <a href="#">gpio_interrupt_mode_t</a> structure that contains the interrupt mode information.

**8.2.6.14** `static void GPIO_SetPinInterruptConfig ( GPIO_Type * base, uint32_t pin,  
gpio_interrupt_mode_t pinInterruptMode ) [inline], [static]`

**8.2.6.15** `static void GPIO_PortEnableInterrupts ( GPIO_Type * base, uint32_t mask )`  
`[inline], [static]`

## GPIO Driver

### Parameters

<i>base</i>	GPIO base pointer.
<i>mask</i>	GPIO pin number macro.

**8.2.6.16** `static void GPIO_EnableInterrupts ( GPIO_Type * base, uint32_t mask )  
[inline], [static]`

### Parameters

<i>base</i>	GPIO base pointer.
<i>mask</i>	GPIO pin number macro.

**8.2.6.17** `static void GPIO_PortDisableInterrupts ( GPIO_Type * base, uint32_t mask )  
[inline], [static]`

### Parameters

<i>base</i>	GPIO base pointer.
<i>mask</i>	GPIO pin number macro.

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

**8.2.6.19** `static uint32_t GPIO_PortGetInterruptFlags ( GPIO_Type * base ) [inline],  
[static]`

### Parameters

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

### Return values

<i>current</i>	pin interrupt status flag.
----------------	----------------------------

**8.2.6.20** `static uint32_t GPIO_GetPinsInterruptFlags ( GPIO_Type * base ) [inline],  
[static]`



## Parameters

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

## Return values

<i>current</i>	pin interrupt status flag.
----------------	----------------------------

### 8.2.6.21 static void GPIO\_PortClearInterruptFlags ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Status flags are cleared by writing a 1 to the corresponding bit position.

## Parameters

<i>base</i>	GPIO base pointer.
<i>mask</i>	GPIO pin number macro.

### 8.2.6.22 static void GPIO\_ClearPinsInterruptFlags ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Status flags are cleared by writing a 1 to the corresponding bit position.

## Parameters

<i>base</i>	GPIO base pointer.
<i>mask</i>	GPIO pin number macro.





## Chapter 9

### I2C: Inter-Integrated Circuit Driver

#### 9.1 Overview

##### Modules

- [I2C Driver](#)
- [I2C FreeRTOS Driver](#)

## 9.2 I2C Driver

### 9.2.1 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.

### 9.2.2 Typical use case

#### 9.2.2.1 Master Operation in functional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 9.2.2.2 Master Operation in interrupt transactional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 9.2.2.3 Slave Operation in functional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 9.2.2.4 Slave Operation in interrupt transactional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

## Data Structures

- struct [i2c\\_master\\_config\\_t](#)

- *I2C master user configuration. [More...](#)*
- struct [i2c\\_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...](#)*

## Macros

- #define [I2C\\_WAIT\\_TIMEOUT](#) 0U /\* Define to zero means keep waiting until the flag is asserted/deassert. \*/  
*Timeout times for waiting flag.*

## 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

- I2C peripheral flags.*
  - enum `_i2c_interrupt_enable` { `kI2C_GlobalInterruptEnable` = `I2C_I2CR_IEN_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, 4))  
*I2C driver version 2.0.4.*

## Initialization and deinitialization

- void `I2C_MasterInit` (`I2C_Type` \*base, const `i2c_master_config_t` \*masterConfig, uint32\_t src-Clock\_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.*
- status\_t [I2C\\_SlaveReadBlocking](#) (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize)  
*Performs a polling receive transaction on the I2C bus.*
- status\_t [I2C\\_MasterTransferBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master polling transfer on the I2C bus.*

## Transactional

- void [I2C\\_MasterTransferCreateHandle](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- status\_t [I2C\\_MasterTransferNonBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)

## 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.*  
• status\_t [I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
• *Aborts an interrupt non-blocking transfer early.*  
• void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
• *Master interrupt handler.*  
• void [I2C\\_SlaveTransferCreateHandle](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, [i2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
• *Initializes the I2C handle which is used in transactional functions.*  
• status\_t [I2C\\_SlaveTransferNonBlocking](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
• *Starts accepting slave transfers.*  
• void [I2C\\_SlaveTransferAbort](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)  
• *Aborts the slave transfer.*  
• status\_t [I2C\\_SlaveTransferGetCount](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, size\_t \*count)  
• *Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.*  
• void [I2C\\_SlaveTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
• *Slave interrupt handler.*

### 9.2.3 Data Structure Documentation

#### 9.2.3.1 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.*

##### 9.2.3.1.0.4 Field Documentation

###### 9.2.3.1.0.4.1 bool i2c\_master\_config\_t::enableMaster

###### 9.2.3.1.0.4.2 uint32\_t i2c\_master\_config\_t::baudRate\_Bps

#### 9.2.3.2 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](#)



- *A sub address.*
- `uint8_t subaddressSize`  
*A size of the command buffer.*
- `uint8_t *volatile data`  
*A transfer buffer.*
- `volatile size_t dataSize`  
*A transfer size.*

### 9.2.3.2.0.5 Field Documentation

#### 9.2.3.2.0.5.1 `uint32_t i2c_master_transfer_t::flags`

#### 9.2.3.2.0.5.2 `uint8_t i2c_master_transfer_t::slaveAddress`

#### 9.2.3.2.0.5.3 `i2c_direction_t i2c_master_transfer_t::direction`

#### 9.2.3.2.0.5.4 `uint32_t i2c_master_transfer_t::subaddress`

Transferred MSB first.

#### 9.2.3.2.0.5.5 `uint8_t i2c_master_transfer_t::subaddressSize`

#### 9.2.3.2.0.5.6 `uint8_t* volatile i2c_master_transfer_t::data`

#### 9.2.3.2.0.5.7 `volatile size_t i2c_master_transfer_t::dataSize`

### 9.2.3.3 `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

### 9.2.3.3.0.6 Field Documentation

9.2.3.3.0.6.1 `i2c_master_transfer_t i2c_master_handle_t::transfer`

9.2.3.3.0.6.2 `size_t i2c_master_handle_t::transferSize`

9.2.3.3.0.6.3 `uint8_t i2c_master_handle_t::state`

9.2.3.3.0.6.4 `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`

9.2.3.3.0.6.5 `void* i2c_master_handle_t::userData`

### 9.2.3.4 struct `i2c_slave_config_t`

#### Data Fields

- `bool enableSlave`  
*Enables the I2C peripheral at initialization time.*
- `uint16_t slaveAddress`  
*A slave address configuration.*

### 9.2.3.4.0.7 Field Documentation

9.2.3.4.0.7.1 `bool i2c_slave_config_t::enableSlave`

9.2.3.4.0.7.2 `uint16_t i2c_slave_config_t::slaveAddress`

### 9.2.3.5 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.*

### 9.2.3.5.0.8 Field Documentation

9.2.3.5.0.8.1 `i2c_slave_transfer_event_t i2c_slave_transfer_t::event`

9.2.3.5.0.8.2 `uint8_t* volatile i2c_slave_transfer_t::data`

9.2.3.5.0.8.3 `volatile size_t i2c_slave_transfer_t::dataSize`

9.2.3.5.0.8.4 `status_t i2c_slave_transfer_t::completionStatus`

Only applies for [kI2C\\_SlaveCompletionEvent](#).

9.2.3.5.0.8.5 `size_t i2c_slave_transfer_t::transferredCount`

### 9.2.3.6 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

### 9.2.3.6.0.9 Field Documentation

9.2.3.6.0.9.1 `volatile uint8_t i2c_slave_handle_t::state`

9.2.3.6.0.9.2 `i2c_slave_transfer_t i2c_slave_handle_t::transfer`

9.2.3.6.0.9.3 `uint32_t i2c_slave_handle_t::eventMask`

9.2.3.6.0.9.4 `i2c_slave_transfer_callback_t i2c_slave_handle_t::callback`

9.2.3.6.0.9.5 `void* i2c_slave_handle_t::userData`

### 9.2.4 Macro Definition Documentation

9.2.4.1 `#define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

9.2.4.2 `#define I2C_WAIT_TIMEOUT 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

### 9.2.5 Typedef Documentation

9.2.5.1 `typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *userData)`

9.2.5.2 `typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, i2c_slave_transfer_t *xfer, void *userData)`

### 9.2.6 Enumeration Type Documentation

#### 9.2.6.1 `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* Timeout polling status flags.

*kStatus\_I2C\_Addr\_Nak* NAK received during the address probe.

#### 9.2.6.2 `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.

### 9.2.6.3 enum \_i2c\_interrupt\_enable

#### Enumerator

***kI2C\_GlobalInterruptEnable*** I2C global interrupt.

### 9.2.6.4 enum i2c\_direction\_t

#### Enumerator

***kI2C\_Write*** Master transmits to the slave.  
***kI2C\_Read*** Master receives from the slave.

### 9.2.6.5 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, only support write only or write+read with no start flag, do not support read only with no start flag.  
***kI2C\_TransferRepeatedStartFlag*** A transfer starts with a repeated start signal.  
***kI2C\_TransferNoStopFlag*** A transfer ends without a stop signal.

## I2C Driver

### 9.2.6.6 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.

#### 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.

## 9.2.7 Function Documentation

### 9.2.7.1 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

**9.2.7.2 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
-------------	------------------

**9.2.7.3 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.
---------------------	--

**9.2.7.4 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);
*
```

## I2C Driver

### Parameters

<i>base</i>	I2C base pointer
<i>slaveConfig</i>	A pointer to the slave configuration structure
<i>srcClock_Hz</i>	I2C peripheral clock frequency in Hz

### 9.2.7.5 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
-------------	------------------

### 9.2.7.6 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.
--------------------	---

### 9.2.7.7 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.

### 9.2.7.8 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.

#### 9.2.7.9 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>	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_ArbitrationLostFlag</li> <li>• kI2C_IntPendingFlagFlag</li> </ul>

#### 9.2.7.10 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.

#### 9.2.7.11 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 <code>i2c_status_flag_t</code> . The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• <code>kI2C_IntPendingFlagFlag</code></li></ul>

### 9.2.7.12 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>• <code>kI2C_GlobalInterruptEnable</code></li><li>• <code>kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</code></li><li>• <code>kI2C_SdaTimeoutInterruptEnable</code></li></ul>

### 9.2.7.13 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>• <code>kI2C_GlobalInterruptEnable</code></li><li>• <code>kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</code></li><li>• <code>kI2C_SdaTimeoutInterruptEnable</code></li></ul>

### 9.2.7.14 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

#### 9.2.7.15 **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.

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.

#### 9.2.7.16 **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.

#### 9.2.7.17 **status\_t I2C\_MasterRepeatedStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )**

Parameters

<i>base</i>	I2C peripheral base pointer
-------------	-----------------------------

## I2C Driver

<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.

### 9.2.7.18 **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_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

### 9.2.7.19 **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_Transfer-DefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

## Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Timeout</i>	Send stop signal failed, timeout.

### 9.2.7.20 **status\_t I2C\_SlaveWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize* )**

## Parameters

<i>base</i>	The I2C peripheral base pointer.
<i>txBuff</i>	The pointer to the data to be transferred.
<i>txSize</i>	The length in bytes of the data to be transferred.

## Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

### 9.2.7.21 **status\_t I2C\_SlaveReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize* )**

## Parameters

## I2C Driver

<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.

### 9.2.7.22 **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.

### 9.2.7.23 **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 i2c_master_handle_t structure to store the transfer state.
<i>callback</i>	pointer to user callback function.

<i>userData</i>	user parameter passed to the callback function.
-----------------	---

#### 9.2.7.24 **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 i2c_master_handle_t structure which stores the transfer state.
<i>xfer</i>	pointer to <a href="#">i2c_master_transfer_t</a> structure.

##### Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.
<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.

#### 9.2.7.25 **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 i2c_master_handle_t structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

##### Return values

---

## I2C Driver

<i>kStatus_InvalidArgument</i>	count is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

### 9.2.7.26 **status\_t I2C\_MasterTransferAbort ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle* )**

#### Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

#### Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_master_handle_t structure which stores the transfer state

#### Return values

<i>kStatus_I2C_Timeout</i>	Timeout during polling flag.
<i>kStatus_Success</i>	Successfully abort the transfer.

### 9.2.7.27 **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.

### 9.2.7.28 **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 <code>i2c_slave_handle_t</code> structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

#### 9.2.7.29 **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 [kI2C\\_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 <code>i2c_slave_handle_t</code> structure which stores the transfer state.
<i>eventMask</i>	Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events.

Return values

<a href="#">#kStatus_Success</a>	Slave transfers were successfully started.
<a href="#">kStatus_I2C_Busy</a>	Slave transfers have already been started on this handle.

#### 9.2.7.30 **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.

## I2C Driver

### Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure which stores the transfer state.

### 9.2.7.31 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

<i>kStatus_InvalidArgument</i>	count is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

### 9.2.7.32 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

## 9.3 I2C FreeRTOS Driver

### 9.3.1 Overview

#### Driver version

- #define **FSL\_I2C\_FREERTOS\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 3))  
*I2C freertos driver version 2.0.3.*

### 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.*

### 9.3.2 Macro Definition Documentation

#### 9.3.2.1 #define FSL\_I2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

### 9.3.3 Function Documentation

#### 9.3.3.1 status\_t I2C\_RTOS\_Init ( i2c\_rtos\_handle\_t \* *handle*, I2C\_Type \* *base*, const i2c\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )

This function initializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the I2C instance to initialize.
<i>masterConfig</i>	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.

## I2C FreeRTOS Driver

### 9.3.3.2 `status_t I2C_RTOS_Deinit ( i2c_rtos_handle_t * handle )`

This function deinitializes the I2C module and the related RTOS context.

## Parameters

<i>handle</i>	The RTOS I2C handle.
---------------	----------------------

### 9.3.3.3 **status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )**

This function performs the I2C transfer according to the data given in the transfer structure.

## Parameters

<i>handle</i>	The RTOS I2C handle.
<i>transfer</i>	A structure specifying the transfer parameters.

## Returns

status of the operation.





## Chapter 10

### PWM: Pulse Width Modulation Driver

#### 10.1 Overview

##### Modules

- [PWM Driver](#)

## 10.2 PWM Driver

### 10.2.1 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.

### 10.2.2 Typical use case

#### 10.2.2.1 PWM output

Output PWM signal on PWM3 module with different dutycycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pwm`

## Enumerations

- enum `pwm_clock_source_t` {  
    `kPWM_PeripheralClock` = 1U,  
    `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.*
- enum `pwm_output_configuration_t` {  
    `kPWM_SetAtRolloverAndClearAtcomparison` = 0U,  
    `kPWM_ClearAtRolloverAndSetAtcomparison`,  
    `kPWM_NoConfigure` }



*PWM Output Configuration.*

- enum `pwm_sample_repeat_t` {  
`kPWM_EachSampleOnce` = 0u,  
`kPWM_EachSampletwice`,  
`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)  
*Software 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)

## PWM Driver

- *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.*

## 10.2.3 Enumeration Type Documentation

### 10.2.3.1 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.

### 10.2.3.2 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.

### 10.2.3.3 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

### 10.2.3.4 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.

## PWM Driver

### 10.2.3.5 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.

### 10.2.3.6 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.

### 10.2.3.7 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.

### 10.2.3.8 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.

### 10.2.3.9 enum pwm\_fifo\_available\_t

Enumerator

- kPWM\_NoDataInFIFOFlag* No data available.

***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

## 10.2.4 Function Documentation

### 10.2.4.1 **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.

### 10.2.4.2 **void PWM\_Deinit ( PWM\_Type \* *base* )**

Parameters

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

### 10.2.4.3 **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;
*

```

## PWM Driver

### Parameters

<i>config</i>	Pointer to user's PWM config structure.
---------------	---

#### 10.2.4.4 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
-------------	-----------------------------

#### 10.2.4.5 static void PWM\_StopTimer ( PWM\_Type \* *base* ) [inline], [static]

### Parameters

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

#### 10.2.4.6 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
-------------	-----------------------------

#### 10.2.4.7 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>

**10.2.4.8** `static void PWM_DisableInterrupts ( PWM_Type * base, uint32_t mask )`  
`[inline], [static]`

## PWM Driver

### 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>

#### 10.2.4.9 **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](#)

#### 10.2.4.10 **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](#)

#### 10.2.4.11 **static void PWM\_clearStatusFlags ( PWM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

### Parameters

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



<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>
-------------	--

#### 10.2.4.12 **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](#)

#### 10.2.4.13 **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.

#### 10.2.4.14 **static uint32\_t PWM\_GetSampleValue ( PWM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The sample value. It can be read only when the PWM is enable.

#### 10.2.4.15 **static void PWM\_SetPeriodValue ( PWM\_Type \* *base*, uint32\_t *value* ) [inline], [static]**

## PWM Driver

### Parameters

<i>base</i>	PWM peripheral base address
<i>mask</i>	The period value. The PWM period register (PWM_PWMPR) determines the period of the PWM output signal. Writing 0xFFFF to this register will achieve the same result as writing 0xFFFE. $PWMO\text{ (Hz)} = PCLK\text{(Hz)} / (\text{period} + 2)$

#### 10.2.4.16 `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\_PWMPR) determines the period of the PWM output signal.

#### 10.2.4.17 `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 11

# UART: Universal Asynchronous Receiver/Transmitter Driver

### 11.1 Overview

#### Modules

- [UART Driver](#)
- [UART FreeRTOS Driver](#)

### 11.2 UART Driver

#### 11.2.1 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 the purpose of optimization/customization. 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_RxRingBufferOverflow`. In the callback function, the upper layer reads data out from the ring buffer. If not, existing data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code.

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/uart`. In this example, the buffer size is 32, but only 31 bytes are used for saving data.

#### 11.2.2 Typical use case

##### 11.2.2.1 UART Send/receive using a polling method

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

### 11.2.2.2 UART Send/receive using an interrupt method

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

### 11.2.2.3 UART Receive using the ringbuffer feature

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

### 11.2.2.4 UART automatic baud rate detect feature

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

## Data Structures

- struct [uart\\_config\\_t](#)  
*UART configuration structure. [More...](#)*
- struct [uart\\_transfer\\_t](#)  
*UART transfer structure. [More...](#)*
- struct [uart\\_handle\\_t](#)  
*UART handle structure. [More...](#)*

## 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_RxRingBufferOverflow` = MAKE\_STATUS(kStatusGroup\_IUART, 8),  
    `kStatus_UART_RxHardwareOverflow` = 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_idle_condition_t` {  
    `kUART_IdleFor4Frames` = 0x0U,  
    `kUART_IdleFor8Frames` = 0x1U,  
    `kUART_IdleFor16Frames` = 0x2U,  
    `kUART_IdleFor32Frames` = 0x3U }  
    *UART idle condition detect.*
- enum `_uart_interrupt_enable`  
    *This structure contains the settings for all of the UART interrupt configurations.*
- enum `_uart_flags` {

```

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_tAirWakeFlag = 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.*

## Functions

- `uint32_t UART_GetInstance (UART_Type *base)`  
*Get the UART instance from peripheral base address.*

### 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.*

## 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\_SetIdleCondition** (UART\_Type \*base, **uart\_idle\_condition\_t** condition)  
*This function is used to configure the IDLE line condition.*
- 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.*

**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.*

**11.2.3 Data Structure Documentation****11.2.3.1 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.*

### 11.2.3.2 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.*

### 11.2.3.3 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.*

## 11.2.4 Macro Definition Documentation

### 11.2.4.1 #define FSL\_UART\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 11.2.5 Typedef Documentation

### 11.2.5.1 typedef void(\* uart\_transfer\_callback\_t)(UART\_Type \*base, uart\_handle\_t \*handle, status\_t status, void \*userData)

## 11.2.6 Enumeration Type Documentation

### 11.2.6.1 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\_RxRingBufferOverflow* UART RX software ring buffer overrun.  
*kStatus\_UART\_RxHardwareOverflow* 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.

### 11.2.6.2 enum uart\_data\_bits\_t

Enumerator

*kUART\_SevenDataBits* Seven data bit.  
*kUART\_EightDataBits* Eight data bit.

### 11.2.6.3 enum uart\_parity\_mode\_t

Enumerator

*kUART\_ParityDisabled* Parity disabled.

## UART Driver

*kUART\_ParityEven* Even error check is selected.

*kUART\_ParityOdd* Odd error check is selected.

### 11.2.6.4 enum uart\_stop\_bit\_count\_t

Enumerator

*kUART\_OneStopBit* One stop bit.

*kUART\_TwoStopBit* Two stop bits.

### 11.2.6.5 enum uart\_idle\_condition\_t

Enumerator

*kUART\_IdleFor4Frames* Idle for more than 4 frames.

*kUART\_IdleFor8Frames* Idle for more than 8 frames.

*kUART\_IdleFor16Frames* Idle for more than 16 frames.

*kUART\_IdleFor32Frames* Idle for more than 32 frames.

### 11.2.6.6 enum \_uart\_interrupt\_enable

### 11.2.6.7 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.  
*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.

## 11.2.7 Function Documentation

### 11.2.7.1 uint32\_t UART\_GetInstance ( UART\_Type \* *base* )

Parameters

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

Returns

UART instance.

### 11.2.7.2 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, UBM, UBRC, URXD, UTXD and UTS[6-3]

Parameters

## UART Driver

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

### 11.2.7.3 **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
------------------------	-------------------------

### 11.2.7.4 **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.
-------------	-------------------------------

### 11.2.7.5 **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 = false; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

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

#### 11.2.7.6 **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, 200000000U);
*
```

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.

#### 11.2.7.7 **static void UART\_Enable ( UART\_Type \* *base* ) [inline], [static]**

Parameters

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

#### 11.2.7.8 **static void UART\_SetIdleCondition ( UART\_Type \* *base*, uart\_idle\_condition\_t *condition* ) [inline], [static]**

## UART Driver

### Parameters

<i>base</i>	UART base pointer.
<i>condition</i>	IDLE line detect condition of the enumerators in <code>_uart_idle_condition</code> .

### 11.2.7.9 `static void UART_Disable ( UART_Type * base ) [inline], [static]`

### Parameters

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

### 11.2.7.10 `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.
----------------	-------------------------------------

### 11.2.7.11 `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.

### 11.2.7.12 `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> .

### 11.2.7.13 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_EnableInterrupts(UART1, kUART_TxEmptyEnable | kUART_RxDataReadyEnable);
*
```

## Parameters

<i>base</i>	UART peripheral base address.
<i>mask</i>	The interrupts to disable. Logical OR of <a href="#">_uart_interrupt_enable</a> .

### 11.2.7.14 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](#).

## UART Driver

**11.2.7.15** `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.

#### 11.2.7.16 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.

#### 11.2.7.17 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.

#### 11.2.7.18 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.

## UART Driver

### 11.2.7.19 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.

#### 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.

### 11.2.7.20 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.

#### 11.2.7.21 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.

#### 11.2.7.22 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.

### 11.2.7.23 void UART\_TransferStopRingBuffer ( UART\_Type \* *base*, uart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.



## Parameters

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

**11.2.7.24 size\_t UART\_TransferGetRxRingBufferLength ( uart\_handle\_t \* *handle* )**

## Parameters

<i>handle</i>	UART handle pointer.
---------------	----------------------

## Returns

Length of received data in RX ring buffer.

**11.2.7.25 status\_t UART\_TransferSendNonBlocking ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uart\_transfer\_t \* *xfer* )**

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the [kStatus\\_UART\\_TxIdle](#) as status parameter.

## Note

The [kStatus\\_UART\\_TxIdle](#) is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the [kUART\\_TransmissionCompleteFlag](#) to ensure that the TX is finished.

## Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure. See <a href="#">uart_transfer_t</a> .

## Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_UART_TxBusy</i>	Previous transmission still not finished; data not all written to TX register yet.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 11.2.7.26 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.

### 11.2.7.27 **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_NoTransferInProgress</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> ;

### 11.2.7.28 **status\_t UART\_TransferReceiveNonBlocking ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )**

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter [kStatus\\_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.

## UART Driver

### 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.

#### 11.2.7.29 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.

#### 11.2.7.30 status\_t UART\_TransferGetReceiveCount ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

### Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>count</i>	Receive bytes count.

Return values

<i>kStatus_NoTransferInProgress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter count;

#### 11.2.7.31 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.

#### 11.2.7.32 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.

#### 11.2.7.33 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 Rx FIFO. The fill level in the Rx FIFO at which a DMA request is generated is controlled by the RXTL bits .

Parameters

## UART Driver

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

### 11.2.7.34 static void UART\_SetTxFifoWatermark ( UART\_Type \* *base*, uint8\_t *watermark* ) [inline], [static]

A maskable interrupt is generated whenever the data level in the TxFIFO falls below the Tx FIFO watermark.

#### Parameters

<i>base</i>	UART base pointer.
<i>watermark</i>	The Tx FIFO watermark.

### 11.2.7.35 static void UART\_SetRxFifoWatermark ( UART\_Type \* *base*, uint8\_t *watermark* ) [inline], [static]

A maskable interrupt is generated whenever the data level in the RxFIFO reaches the Rx FIFO watermark.

#### Parameters

<i>base</i>	UART base pointer.
<i>watermark</i>	The Rx FIFO watermark.

### 11.2.7.36 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>

### 11.2.7.37 static bool UART\_IsAutoBaudRateComplete ( UART\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

- true: Automatic baud rate detection has finished.
  - false: Automatic baud rate detection has not finished.

### 11.2.8 Variable Documentation

- 11.2.8.1 `uint32_t uart_config_t::baudRate_Bps`
- 11.2.8.2 `uart_parity_mode_t uart_config_t::parityMode`
- 11.2.8.3 `uart_stop_bit_count_t uart_config_t::stopBitCount`
- 11.2.8.4 `uint8_t* uart_transfer_t::data`
- 11.2.8.5 `size_t uart_transfer_t::dataSize`
- 11.2.8.6 `uint8_t* volatile uart_handle_t::txData`
- 11.2.8.7 `volatile size_t uart_handle_t::txDataSize`
- 11.2.8.8 `size_t uart_handle_t::txDataSizeAll`
- 11.2.8.9 `uint8_t* volatile uart_handle_t::rxData`
- 11.2.8.10 `volatile size_t uart_handle_t::rxDataSize`
- 11.2.8.11 `size_t uart_handle_t::rxDataSizeAll`
- 11.2.8.12 `uint8_t* uart_handle_t::rxRingBuffer`
- 11.2.8.13 `size_t uart_handle_t::rxRingBufferSize`
- 11.2.8.14 `volatile uint16_t uart_handle_t::rxRingBufferHead`
- 11.2.8.15 `volatile uint16_t uart_handle_t::rxRingBufferTail`
- 11.2.8.16 `uart_transfer_callback_t uart_handle_t::callback`
- 11.2.8.17 `void* uart_handle_t::userData`
- 11.2.8.18 `volatile uint8_t uart_handle_t::txState`



## 11.3 UART FreeRTOS Driver

### 11.3.1 Overview

#### Data Structures

- struct `uart_rtos_config_t`  
*UART configuration structure. [More...](#)*

#### Driver version

- #define `FSL_UART_FREERTOS_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*UART freertos driver version 2.0.0.*

#### 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.*

### 11.3.2 Data Structure Documentation

#### 11.3.2.1 struct `uart_rtos_config_t`

##### Data Fields

- UART\_Type \* `base`  
*UART base address.*
- uint32\_t `srcclk`  
*UART source clock in Hz.*
- uint32\_t `baudrate`  
*Desired communication speed.*
- `uart_parity_mode_t` `parity`  
*Parity setting.*

## UART FreeRTOS Driver

- `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.*

### 11.3.3 Macro Definition Documentation

#### 11.3.3.1 `#define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

### 11.3.4 Function Documentation

#### 11.3.4.1 `int UART_RTOS_Init ( uart_rtos_handle_t * handle, uart_handle_t * t_handle, const uart_rtos_config_t * cfg )`

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.

#### 11.3.4.2 `int UART_RTOS_Deinit ( uart_rtos_handle_t * handle )`

This function deinitializes the UART module, sets all register values to reset value, and frees the resources.

Parameters

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

#### 11.3.4.3 `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.

#### 11.3.4.4 int UART\_RTOS\_Receive ( uart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from UART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

## Parameters

<i>handle</i>	The RTOS UART handle.
<i>buffer</i>	The pointer to the buffer to write received data.
<i>length</i>	The number of bytes to receive.
<i>received</i>	The pointer to a variable of size_t where the number of received data is filled.



## Chapter 12

### MU: Messaging Unit

#### 12.1 Overview

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

#### 12.2 Function description

The MU driver provides these functions:

- Functions to initialize the MU module.
- Functions to send and receive messages.
- Functions for MU flags for both MU sides.
- Functions for status flags and interrupts.
- Other miscellaneous functions.

##### 12.2.1 MU initialization

The function [MU\\_Init\(\)](#) initializes the MU module and enables the MU clock. It should be called before any other MU functions.

The function [MU\\_Deinit\(\)](#) deinitializes the MU module and disables the MU clock. No MU functions can be called after this function.

##### 12.2.2 MU message

The MU message must be sent when the transmit register is empty. The MU driver provides blocking API and non-blocking API to send message.

The [MU\\_SendMsgNonBlocking\(\)](#) function writes a message to the MU transmit register without checking the transmit register status. The upper layer should check that the transmit register is empty before calling this function. This function can be used in the ISR for better performance.

The [MU\\_SendMsg\(\)](#) function is a blocking function. It waits until the transmit register is empty and sends the message.

Correspondingly, there are blocking and non-blocking APIs for receiving a message. The [MU\\_ReadMsgNonBlocking\(\)](#) function is a non-blocking API. The [MU\\_ReadMsg\(\)](#) function is the blocking API.

## Function description

### 12.2.3 MU flags

The MU driver provides 3-bit general purpose flags. When the flags are set on one side, they are reflected on the other side.

The MU flags must be set when the previous flags have been updated to the other side. The MU driver provides a non-blocking function and a blocking function. The blocking function [MU\\_SetFlags\(\)](#) waits until previous flags have been updated to the other side and then sets flags. The non-blocking function sets the flags directly. Ensure that the `kMU_FlagsUpdatingFlag` is not pending before calling this function.

The function [MU\\_GetFlags\(\)](#) gets the MU flags on the current side.

### 12.2.4 Status and interrupt

The function [MU\\_GetStatusFlags\(\)](#) returns all MU status flags. Use the `_mu_status_flags` to check for specific flags, for example, to check RX0 and RX1 register full, use the following code:

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/mu`. The receive full flags are cleared automatically after messages are read out. The transmit empty flags are cleared automatically after new messages are written to the transmit register. The general purpose interrupt flags must be cleared manually using the function [MU\\_ClearStatusFlags\(\)](#).

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/mu`. To enable or disable a specific interrupt, use [MU\\_EnableInterrupts\(\)](#) and [MU\\_DisableInterrupts\(\)](#) functions. The interrupts to enable or disable should be passed in as a bit mask of the `_mu_interrupt_enable`.

The [MU\\_TriggerInterrupts\(\)](#) function triggers general purpose interrupts and NMI to the other core. The interrupts to trigger are passed in as a bit mask of the `_mu_interrupt_trigger`. If previously triggered interrupts have not been processed by the other side, this function returns an error.

### 12.2.5 MU misc functions

The [MU\\_BootCoreB\(\)](#) and [MU\\_HoldCoreBReset\(\)](#) functions should only be used from A side. They are used to boot the core B or to hold core B in reset.

The [MU\\_ResetBothSides\(\)](#) function resets MU at both A and B sides. However, only the A side can call this function.

If a core enters stop mode, the platform clock of this core is disabled by default. The function [MU\\_SetClockOnOtherCoreEnable\(\)](#) forces the other core's platform clock to remain enabled even after that core has entered a stop mode. In this case, the other core's platform clock keeps running until the current core enters stop mode too.

Function [MU\\_GetOtherCorePowerMode\(\)](#) gets the power mode of the other core.

## Enumerations

- enum `_mu_status_flags` {  
`kMU_Tx0EmptyFlag` = (1U << (MU\_SR\_TEn\_SHIFT + 3U)),  
`kMU_Tx1EmptyFlag` = (1U << (MU\_SR\_TEn\_SHIFT + 2U)),  
`kMU_Tx2EmptyFlag` = (1U << (MU\_SR\_TEn\_SHIFT + 1U)),  
`kMU_Tx3EmptyFlag` = (1U << (MU\_SR\_TEn\_SHIFT + 0U)),  
`kMU_Rx0FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 3U)),  
`kMU_Rx1FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 2U)),  
`kMU_Rx2FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 1U)),  
`kMU_Rx3FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 0U)),  
`kMU_GenInt0Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 3U)),  
`kMU_GenInt1Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 2U)),  
`kMU_GenInt2Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 1U)),  
`kMU_GenInt3Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 0U)),  
`kMU_EventPendingFlag` = MU\_SR\_EP\_MASK,  
`kMU_FlagsUpdatingFlag` = MU\_SR\_FUP\_MASK }  
*MU status flags.*
- enum `_mu_interrupt_enable` {  
`kMU_Tx0EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 3U)),  
`kMU_Tx1EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 2U)),  
`kMU_Tx2EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 1U)),  
`kMU_Tx3EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 0U)),  
`kMU_Rx0FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 3U)),  
`kMU_Rx1FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 2U)),  
`kMU_Rx2FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 1U)),  
`kMU_Rx3FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 0U)),  
`kMU_GenInt0InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 3U)),  
`kMU_GenInt1InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 2U)),  
`kMU_GenInt2InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 1U)),  
`kMU_GenInt3InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 0U)) }  
*MU interrupt source to enable.*
- enum `_mu_interrupt_trigger` {  
`kMU_NmiInterruptTrigger` = MU\_CR\_NMI\_MASK,  
`kMU_GenInt0InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 3U)),  
`kMU_GenInt1InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 2U)),  
`kMU_GenInt2InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 1U)),  
`kMU_GenInt3InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 0U)) }  
*MU interrupt that could be triggered to the other core.*

## Driver version

- #define `FSL_MU_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*MU driver version 2.0.2.*

## Function description

### MU initialization.

- void [MU\\_Init](#) (MU\_Type \*base)  
*Initializes the MU module.*
- void [MU\\_Deinit](#) (MU\_Type \*base)  
*De-initializes the MU module.*

### MU Message

- static void [MU\\_SendMsgNonBlocking](#) (MU\_Type \*base, uint32\_t regIndex, uint32\_t msg)  
*Writes a message to the TX register.*
- void [MU\\_SendMsg](#) (MU\_Type \*base, uint32\_t regIndex, uint32\_t msg)  
*Blocks to send a message.*
- static uint32\_t [MU\\_ReceiveMsgNonBlocking](#) (MU\_Type \*base, uint32\_t regIndex)  
*Reads a message from the RX register.*
- uint32\_t [MU\\_ReceiveMsg](#) (MU\_Type \*base, uint32\_t regIndex)  
*Blocks to receive a message.*

### MU Flags

- static void [MU\\_SetFlagsNonBlocking](#) (MU\_Type \*base, uint32\_t flags)  
*Sets the 3-bit MU flags reflect on the other MU side.*
- void [MU\\_SetFlags](#) (MU\_Type \*base, uint32\_t flags)  
*Blocks setting the 3-bit MU flags reflect on the other MU side.*
- static uint32\_t [MU\\_GetFlags](#) (MU\_Type \*base)  
*Gets the current value of the 3-bit MU flags set by the other side.*

### Status and Interrupt.

- static uint32\_t [MU\\_GetStatusFlags](#) (MU\_Type \*base)  
*Gets the MU status flags.*
- static void [MU\\_ClearStatusFlags](#) (MU\_Type \*base, uint32\_t mask)  
*Clears the specific MU status flags.*
- static void [MU\\_EnableInterrupts](#) (MU\_Type \*base, uint32\_t mask)  
*Enables the specific MU interrupts.*
- static void [MU\\_DisableInterrupts](#) (MU\_Type \*base, uint32\_t mask)  
*Disables the specific MU interrupts.*
- status\_t [MU\\_TriggerInterrupts](#) (MU\_Type \*base, uint32\_t mask)  
*Triggers interrupts to the other core.*
- static void [MU\\_ClearNmi](#) (MU\_Type \*base)  
*Clear non-maskable interrupt (NMI) sent by the other core.*

### MU misc functions

- void [MU\\_BootCoreB](#) (MU\_Type \*base, mu\_core\_boot\_mode\_t mode)  
*Boots the core at B side.*
- static void [MU\\_HoldCoreBReset](#) (MU\_Type \*base)  
*Holds the core reset of B side.*
- void [MU\\_BootOtherCore](#) (MU\_Type \*base, mu\_core\_boot\_mode\_t mode)  
*Boots the other core.*
- static void [MU\\_HoldOtherCoreReset](#) (MU\_Type \*base)



- Holds the other core reset.*
- static void [MU\\_ResetBothSides](#) (MU\_Type \*base)  
*Resets the MU for both A side and B side.*
- void [MU\\_HardwareResetOtherCore](#) (MU\_Type \*base, bool waitReset, bool holdReset, mu\_core\_-boot\_mode\_t bootMode)  
*Hardware reset the other core.*
- static void [MU\\_SetClockOnOtherCoreEnable](#) (MU\_Type \*base, bool enable)  
*Enables or disables the clock on the other core.*
- static mu\_power\_mode\_t [MU\\_GetOtherCorePowerMode](#) (MU\_Type \*base)  
*Gets the power mode of the other core.*

## 12.3 Macro Definition Documentation

### 12.3.1 #define FSL\_MU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## 12.4 Enumeration Type Documentation

### 12.4.1 enum \_mu\_status\_flags

Enumerator

***kMU\_Tx0EmptyFlag*** TX0 empty.  
***kMU\_Tx1EmptyFlag*** TX1 empty.  
***kMU\_Tx2EmptyFlag*** TX2 empty.  
***kMU\_Tx3EmptyFlag*** TX3 empty.  
***kMU\_Rx0FullFlag*** RX0 full.  
***kMU\_Rx1FullFlag*** RX1 full.  
***kMU\_Rx2FullFlag*** RX2 full.  
***kMU\_Rx3FullFlag*** RX3 full.  
***kMU\_GenInt0Flag*** General purpose interrupt 0 pending.  
***kMU\_GenInt1Flag*** General purpose interrupt 0 pending.  
***kMU\_GenInt2Flag*** General purpose interrupt 0 pending.  
***kMU\_GenInt3Flag*** General purpose interrupt 0 pending.  
***kMU\_EventPendingFlag*** MU event pending.  
***kMU\_FlagsUpdatingFlag*** MU flags update is on-going.

### 12.4.2 enum \_mu\_interrupt\_enable

Enumerator

***kMU\_Tx0EmptyInterruptEnable*** TX0 empty.  
***kMU\_Tx1EmptyInterruptEnable*** TX1 empty.  
***kMU\_Tx2EmptyInterruptEnable*** TX2 empty.  
***kMU\_Tx3EmptyInterruptEnable*** TX3 empty.  
***kMU\_Rx0FullInterruptEnable*** RX0 full.  
***kMU\_Rx1FullInterruptEnable*** RX1 full.

## Function Documentation

***kMU\_Rx2FullInterruptEnable*** RX2 full.  
***kMU\_Rx3FullInterruptEnable*** RX3 full.  
***kMU\_GenInt0InterruptEnable*** General purpose interrupt 0.  
***kMU\_GenInt1InterruptEnable*** General purpose interrupt 1.  
***kMU\_GenInt2InterruptEnable*** General purpose interrupt 2.  
***kMU\_GenInt3InterruptEnable*** General purpose interrupt 3.

### 12.4.3 enum \_mu\_interrupt\_trigger

Enumerator

***kMU\_NmiInterruptTrigger*** NMI interrupt.  
***kMU\_GenInt0InterruptTrigger*** General purpose interrupt 0.  
***kMU\_GenInt1InterruptTrigger*** General purpose interrupt 1.  
***kMU\_GenInt2InterruptTrigger*** General purpose interrupt 2.  
***kMU\_GenInt3InterruptTrigger*** General purpose interrupt 3.

## 12.5 Function Documentation

### 12.5.1 void MU\_Init ( MU\_Type \* *base* )

This function enables the MU clock only.

Parameters

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

### 12.5.2 void MU\_Deinit ( MU\_Type \* *base* )

This function disables the MU clock only.

Parameters

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

### 12.5.3 static void MU\_SendMsgNonBlocking ( MU\_Type \* *base*, uint32\_t *regIndex*, uint32\_t *msg* ) [inline], [static]

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```

* while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } // Wait for
    TX0 register empty.
* MU_SendMsgNonBlocking(base, 0U, MSG_VAL); // Write message to the TX0 register.
*

```

## Parameters

<i>base</i>	MU peripheral base address.
<i>regIndex</i>	TX register index.
<i>msg</i>	Message to send.

#### 12.5.4 void MU\_SendMsg ( MU\_Type \* *base*, uint32\_t *regIndex*, uint32\_t *msg* )

This function waits until the TX register is empty and sends the message.

## Parameters

<i>base</i>	MU peripheral base address.
<i>regIndex</i>	TX register index.
<i>msg</i>	Message to send.

#### 12.5.5 static uint32\_t MU\_ReceiveMsgNonBlocking ( MU\_Type \* *base*, uint32\_t *regIndex* ) [inline], [static]

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```

* uint32_t msg;
* while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
* {
* } // Wait for the RX0 register full.
*
* msg = MU_ReceiveMsgNonBlocking(base, 0U); // Read message from RX0 register.
*

```

## Parameters

## Function Documentation

<i>base</i>	MU peripheral base address.
<i>regIndex</i>	TX register index.

Returns

The received message.

### 12.5.6 uint32\_t MU\_ReceiveMsg ( MU\_Type \* *base*, uint32\_t *regIndex* )

This function waits until the RX register is full and receives the message.

Parameters

<i>base</i>	MU peripheral base address.
<i>regIndex</i>	RX register index.

Returns

The received message.

### 12.5.7 static void MU\_SetFlagsNonBlocking ( MU\_Type \* *base*, uint32\_t *flags* ) [inline], [static]

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU\_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU\_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU\_FlagsUpdatingFlag is cleared before calling this function.

```
* while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
* {
* } // Wait for previous MU flags updating.
*
* MU_SetFlagsNonBlocking(base, 0U); // Set the mU flags.
*
```

Parameters

<i>base</i>	MU peripheral base address.
<i>flags</i>	The 3-bit MU flags to set.

### 12.5.8 void MU\_SetFlags ( MU\_Type \* *base*, uint32\_t *flags* )

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU\_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU\_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU\_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

Parameters

<i>base</i>	MU peripheral base address.
<i>flags</i>	The 3-bit MU flags to set.

### 12.5.9 static uint32\_t MU\_GetFlags ( MU\_Type \* *base* ) [inline], [static]

This function gets the current 3-bit MU flags on the current side.

Parameters

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

Returns

flags Current value of the 3-bit flags.

### 12.5.10 static uint32\_t MU\_GetStatusFlags ( MU\_Type \* *base* ) [inline], [static]

This function returns the bit mask of the MU status flags. See \_mu\_status\_flags.

```
* uint32_t flags;
* flags = MU_GetStatusFlags(base); // Get all status flags.
* if (kMU_Tx0EmptyFlag & flags)
* {
*     // The TX0 register is empty. Message can be sent.
*     MU_SendMsgNonBlocking(base, 0U, MSG0_VAL);
* }
* if (kMU_Tx1EmptyFlag & flags)
* {
```

## Function Documentation

```
* // The TX1 register is empty. Message can be sent.
* MU_SendMsgNonBlocking(base, 1U, MSG1_VAL);
* }
*
```

### Parameters

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

### Returns

Bit mask of the MU status flags, see `_mu_status_flags`.

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

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See `_mu_status_flags`.

```
* //Clear general interrupt 0 and general interrupt 1 pending flags.
* MU_ClearStatusFlags(base, kMU_GenInt0Flag |
*   kMU_GenInt1Flag);
*
```

### Parameters

<i>base</i>	MU peripheral base address.
<i>mask</i>	Bit mask of the MU status flags. See <code>_mu_status_flags</code> . The following flags are cleared by hardware, this function could not clear them. <ul style="list-style-type: none"><li>• kMU_Tx0EmptyFlag</li><li>• kMU_Tx1EmptyFlag</li><li>• kMU_Tx2EmptyFlag</li><li>• kMU_Tx3EmptyFlag</li><li>• kMU_Rx0FullFlag</li><li>• kMU_Rx1FullFlag</li><li>• kMU_Rx2FullFlag</li><li>• kMU_Rx3FullFlag</li><li>• kMU_EventPendingFlag</li><li>• kMU_FlagsUpdatingFlag</li><li>• kMU_OtherSideInResetFlag</li></ul>

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

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See `_mu_interrupt_enable`.

```
* // Enable general interrupt 0 and TX0 empty interrupt.
* MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable |
*   kMU_Tx0EmptyInterruptEnable);
*
```

Parameters

<i>base</i>	MU peripheral base address.
<i>mask</i>	Bit mask of the MU interrupts. See <code>_mu_interrupt_enable</code> .

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

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See `_mu_interrupt_enable`.

```
* // Disable general interrupt 0 and TX0 empty interrupt.
* MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable |
*   kMU_Tx0EmptyInterruptEnable);
*
```

Parameters

<i>base</i>	MU peripheral base address.
<i>mask</i>	Bit mask of the MU interrupts. See <code>_mu_interrupt_enable</code> .

### 12.5.14 status\_t MU\_TriggerInterrupts ( MU\_Type \* *base*, uint32\_t *mask* )

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See `_mu_interrupt_trigger`. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
* if (kStatus_Success != MU_TriggerInterrupts(base,
*   kMU_GenInt0InterruptTrigger |
*   kMU_GenInt2InterruptTrigger))
* {
*   // Previous general purpose interrupt 0 or general purpose interrupt 2
*   // has not been processed by the other core.
* }
*
```

## Function Documentation

### Parameters

<i>base</i>	MU peripheral base address.
<i>mask</i>	Bit mask of the interrupts to trigger. See <code>_mu_interrupt_trigger</code> .

### Return values

<i>kStatus_Success</i>	Interrupts have been triggered successfully.
<i>kStatus_Fail</i>	Previous interrupts have not been accepted.

### 12.5.15 `static void MU_ClearNmi ( MU_Type * base ) [inline], [static]`

This function clears non-maskable interrupt (NMI) sent by the other core.

#### Parameters

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

### 12.5.16 `void MU_BootCoreB ( MU_Type * base, mu_core_boot_mode_t mode )`

This function sets the B side core's boot configuration and releases the core from reset.

#### Parameters

<i>base</i>	MU peripheral base address.
<i>mode</i>	Core B boot mode.

#### Note

Only MU side A can use this function.

### 12.5.17 `static void MU_HoldCoreBReset ( MU_Type * base ) [inline], [static]`

This function causes the core of B side to be held in reset following any reset event.



## Parameters

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

## Note

Only A side could call this function.

### 12.5.18 void MU\_BootOtherCore ( MU\_Type \* *base*, mu\_core\_boot\_mode\_t *mode* )

This function boots the other core with a boot configuration.

## Parameters

<i>base</i>	MU peripheral base address.
<i>mode</i>	The other core boot mode.

### 12.5.19 static void MU\_HoldOtherCoreReset ( MU\_Type \* *base* ) [inline], [static]

This function causes the other core to be held in reset following any reset event.

## Parameters

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

### 12.5.20 static void MU\_ResetBothSides ( MU\_Type \* *base* ) [inline], [static]

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

## Parameters

---

## Function Documentation

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

### Note

For some platforms, only MU side A could use this function, check reference manual for details.

### 12.5.21 void MU\_HardwareResetOtherCore ( MU\_Type \* *base*, bool *waitReset*, bool *holdReset*, mu\_core\_boot\_mode\_t *bootMode* )

This function resets the other core, the other core could mask the hardware reset by calling MU\_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU\_BootOtherCore to control the other core reset workflow.

Example 1: Reset the other core, and no hold reset

```
* MU_HardwareResetOtherCore(MU_A, true, false, bootMode);  
*
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later.

```
* // Here the other core enters reset, and the reset is hold  
* MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);  
* // Current core boot the other core when necessary.  
* MU_BootOtherCore(MU_A, bootMode);  
*
```

### Parameters

<i>base</i>	MU peripheral base address.
<i>waitReset</i>	Wait the other core enters reset. <ul style="list-style-type: none"><li>• true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.</li><li>• false: Don't wait the reset.</li></ul>

<i>holdReset</i>	Hold the other core reset or not. <ul style="list-style-type: none"> <li>• true: Hold the other core in reset, this function returns directly when the other core enters reset.</li> <li>• false: Don't hold the other core in reset, this function waits until the other core out of reset.</li> </ul>
<i>bootMode</i>	Boot mode of the other core, if <code>holdReset</code> is true, this parameter is useless.

### 12.5.22 static void MU\_SetClockOnOtherCoreEnable ( MU\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the platform clock on the other core when that core enters a stop mode. If disabled, the platform clock for the other core is disabled when it enters stop mode. If enabled, the platform clock keeps running on the other core in stop mode, until this core also enters stop mode.

Parameters

<i>base</i>	MU peripheral base address.
<i>enable</i>	Enable or disable the clock on the other core.

### 12.5.23 static mu\_power\_mode\_t MU\_GetOtherCorePowerMode ( MU\_Type \* *base* ) [inline], [static]

This function gets the power mode of the other core.

Parameters

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

Returns

Power mode of the other core.



## Chapter 13

# QSPI: Quad Serial Peripheral Interface Driver

### 13.1 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.

### 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)`  
*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 endianness.*
- enum `_qspi_error_flags` {  
`kQSPI_DataLearningFail` = (int)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` = (int)0x8C83F8D1U }  
*QSPI error flags.*
- enum `_qspi_flags` {

```

kQSPI_DataLearningSamplePoint = (int)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_AHBCommandPriorityGranted = 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 = (int)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 = (int)0x8C83F8D1U }
    QSPI interrupt enable.
• enum _qspi_dma_enable { kQSPI_RxBufferDrainDMAEnable = QuadSPI_RSER_RBDDE_MAS-
    K }
    QSPI DMA request flag.
• enum qspi_dqs_phrase_shift_t {
    kQSPI_DQSNoPhraseShift = 0x0U,
    kQSPI_DQSPhraseShift45Degree,
    kQSPI_DQSPhraseShift90Degree,
    kQSPI_DQSPhraseShift135Degree }

```

## Overview

*Phrase shift number for DQS mode.*

## Driver version

- #define [FSL\\_QSPI\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 2))  
*QSPI driver version 2.0.2.*

## Initialization and deinitialization

- uint32\_t [QSPI\\_GetInstance](#) (QuadSPI\_Type \*base)  
*Get the instance number for QSPI.*
- 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\\_EnableDDRMMode](#) (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.*

## 13.2 Data Structure Documentation

### 13.2.1 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.*

### 13.2.2 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.*

### 13.2.3 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.*

### 13.2.3.0.0.10 Field Documentation

13.2.3.0.0.10.1 `uint8_t qspi_config_t::rxWatermark`

13.2.3.0.0.10.2 `uint32_t qspi_config_t::AHBbufferSize[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]`

13.2.3.0.0.10.3 `uint8_t qspi_config_t::AHBbufferMaster[FSL_FEATURE_QSPI_AHB_BUFFER_COUNT]`

13.2.3.0.0.10.4 `bool qspi_config_t::enableAHBbuffer3AllMaster`

## 13.2.4 `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 lookupable [FSL_FEATURE_QSPI_LUT_DEPTH]`  
*Flash command in LUT.*
- `uint32_t dataHoldTime`  
*Data line hold time.*
- `uint32_t CSHoldTime`  
*CS line hold time.*
- `uint32_t CSSetupTime`  
*CS line setup time.*
- `uint32_t cloumnspace`  
*Column space size.*
- `uint32_t dataLearnValue`  
*Data Learn value if enable data learn.*
- `qspi_endianness_t endian`  
*Flash data endianess.*
- `bool enableWordAddress`  
*If enable word address.*

## Enumeration Type Documentation

### 13.2.4.0.0.11 Field Documentation

13.2.4.0.0.11.1 `uint32_t qspi_flash_config_t::dataHoldTime`

13.2.4.0.0.11.2 `qspi_endianness_t qspi_flash_config_t::endian`

13.2.4.0.0.11.3 `bool qspi_flash_config_t::enableWordAddress`

### 13.2.5 `struct qspi_transfer_t`

## Data Fields

- `uint32_t * data`  
*Pointer to data to transmit.*
- `size_t dataSize`  
*Bytes to be transmit.*

## 13.3 Macro Definition Documentation

13.3.1 `#define FSL_QSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

## 13.4 Enumeration Type Documentation

### 13.4.1 `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.

### 13.4.2 `enum qspi_read_area_t`

Enumerator

***kQSPI\_ReadAHB*** QSPI read from AHB buffer.  
***kQSPI\_ReadIP*** QSPI read from IP FIFO.

### 13.4.3 `enum qspi_command_seq_t`

Enumerator

***kQSPI\_IPSeq*** IP command sequence.  
***kQSPI\_BufferSeq*** Buffer command sequence.

### 13.4.4 enum qspi\_fifo\_t

Enumerator

*kQSPI\_TxFifo* QSPI Tx FIFO.  
*kQSPI\_RxFifo* QSPI Rx FIFO.  
*kQSPI\_AllFifo* QSPI all FIFO, including Tx and Rx.

### 13.4.5 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

### 13.4.6 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.

### 13.4.7 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\_AHBTransactionPending*** AHB access transaction pending.  
***kQSPI\_AHBCommandPriorityGranted*** AHB command priority granted.  
***kQSPI\_AHBAccess*** AHB access.  
***kQSPI\_IPAccess*** IP access.  
***kQSPI\_Busy*** Module busy.  
***kQSPI\_StateAll*** All flags.

### 13.4.8 enum \_qspi\_interrupt\_enable

#### Enumerator

***kQSPI\_DataLearningFailInterruptEnable*** Data learning pattern failure interrupt enable.  
***kQSPI\_TxBufferFillInterruptEnable*** Tx buffer fill interrupt enable.  
***kQSPI\_TxBufferUnderrunInterruptEnable*** Tx buffer underrun interrupt enable.  
***kQSPI\_IllegalInstructionInterruptEnable*** Illegal instruction error interrupt enable.  
***kQSPI\_RxBufferOverflowInterruptEnable*** Rx buffer overflow interrupt enable.  
***kQSPI\_RxBufferDrainInterruptEnable*** Rx buffer drain interrupt enable.  
***kQSPI\_AHBSequenceErrorInterruptEnable*** AHB sequence error interrupt enable.  
***kQSPI\_AHBBufferOverflowInterruptEnable*** AHB buffer overflow interrupt enable.  
***kQSPI\_IPCommandUsageErrorInterruptEnable*** IP command usage error interrupt enable.  
***kQSPI\_IPCommandTriggerDuringAHBAccessInterruptEnable*** 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.

### 13.4.9 enum \_qspi\_dma\_enable

Enumerator

***kQSPI\_RxBufferDrainDMAEnable*** Rx buffer drain DMA.

### 13.4.10 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.

## 13.5 Function Documentation

### 13.5.1 uint32\_t QSPI\_GetInstance ( QuadSPI\_Type \* *base* )

Parameters

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

### 13.5.2 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.

### 13.5.3 void QSPI\_GetDefaultQspiConfig ( qspi\_config\_t \* *config* )

## Function Documentation

### Parameters

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

### 13.5.4 void QSPI\_Deinit ( QuadSPI\_Type \* *base* )

Clears the QSPI state and QSPI module registers.

### Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

### 13.5.5 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.

### 13.5.6 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.
-------------	--------------------------

### 13.5.7 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.

### 13.5.8 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.

### 13.5.9 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.

### 13.5.10 static void QSPI\_ClearErrorFlag ( QuadSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

<i>base</i>	Pointer to QuadSPI Type.
-------------	--------------------------

## Function Documentation

<i>mask</i>	Which kind of QSPI flags to be cleared, a combination of <code>_qspi_error_flags</code> .
-------------	---

**13.5.11** `static void QSPI_EnableInterrupts ( QuadSPI_Type * base, uint32_t mask ) [inline], [static]`

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>mask</i>	QSPI interrupt source.

**13.5.12** `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.

**13.5.13** `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.

**13.5.14** `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.

### 13.5.15 `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.

### 13.5.16 `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.

### 13.5.17 `static void QSPI_SetIPCommandSize ( QuadSPI_Type * base, uint32_t size ) [inline], [static]`

## Parameters

---

## Function Documentation

<i>base</i>	Pointer to QuadSPI Type.
<i>size</i>	IP command size.

### 13.5.18 void QSPI\_ExecutelPCommand ( 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.

### 13.5.19 void QSPI\_ExecuteAHBCommand ( QuadSPI\_Type \* *base*, uint32\_t *index* )

Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>index</i>	AHB command located in which LUT table index.

### 13.5.20 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.

### 13.5.21 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.

### 13.5.22 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.

### 13.5.23 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.

### 13.5.24 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.

### 13.5.25 static void QSPI\_EnableDDRMMode ( QuadSPI\_Type \* *base*, bool *enable* ) [inline], [static]

## Function Documentation

### Parameters

<i>base</i>	QSPI base pointer
<i>enable</i>	True means enable DDR mode, false means disable DDR mode.

### 13.5.26 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.

### 13.5.27 void QSPI\_WriteBlocking ( QuadSPI\_Type \* *base*, uint32\_t \* *buffer*, size\_t *size* )

### Note

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

### Parameters

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

### 13.5.28 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

### 13.5.29 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

### 13.5.30 uint32\_t QSPI\_ReadData ( QuadSPI\_Type \* *base* )

#### Parameters

<i>base</i>	QSPI base pointer
-------------	-------------------

#### Returns

The data in the FIFO.

### 13.5.31 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.

## Function Documentation

### Parameters

<i>base</i>	Pointer to QuadSPI Type.
<i>xfer</i>	QSPI transfer structure.

### 13.5.32 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.



## Chapter 14

# RDC: Resource Domain Controller

### 14.1 Overview

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

The Resource Domain Controller (RDC) provides robust support for the isolation of destination memory mapped locations such as peripherals and memory to a single core, a bus master, or set of cores and bus masters.

The RDC driver should be used together with the RDC\_SEMA42 driver.

### Data Structures

- struct [rdc\\_hardware\\_config\\_t](#)  
*RDC hardware configuration. [More...](#)*
- struct [rdc\\_domain\\_assignment\\_t](#)  
*Master domain assignment. [More...](#)*
- struct [rdc\\_periph\\_access\\_config\\_t](#)  
*Peripheral domain access permission configuration. [More...](#)*
- struct [rdc\\_mem\\_access\\_config\\_t](#)  
*Memory region domain access control configuration. [More...](#)*
- struct [rdc\\_mem\\_status\\_t](#)  
*Memory region access violation status. [More...](#)*

### Macros

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

### Enumerations

- enum [\\_rdc\\_interrupts](#) { [kRDC\\_RestoreCompleteInterrupt](#) = RDC\_INTCTRL\_RCI\_EN\_MASK }
- enum [\\_rdc\\_flags](#) { [kRDC\\_PowerDownDomainOn](#) = RDC\_STAT\_PDS\_MASK }
- enum [\\_rdc\\_access\\_policy](#) {  
[kRDC\\_NoAccess](#) = 0,  
[kRDC\\_WriteOnly](#) = 1,  
[kRDC\\_ReadOnly](#) = 2,  
[kRDC\\_ReadWrite](#) = 3 }  
*Access permission policy.*

### Functions

- void [RDC\\_Init](#) (RDC\_Type \*base)

## Overview

- Initializes the RDC module.*
- void [RDC\\_Deinit](#) (RDC\_Type \*base)  
*De-initializes the RDC module.*
- static void [RDC\\_GetHardwareConfig](#) (RDC\_Type \*base, [rdc\\_hardware\\_config\\_t](#) \*config)  
*Gets the RDC hardware configuration.*
- static void [RDC\\_EnableInterrupts](#) (RDC\_Type \*base, uint32\_t mask)  
*Enable interrupts.*
- static void [RDC\\_DisableInterrupts](#) (RDC\_Type \*base, uint32\_t mask)  
*Disable interrupts.*
- static uint32\_t [RDC\\_GetInterruptStatus](#) (RDC\_Type \*base)  
*Get the interrupt pending status.*
- static void [RDC\\_ClearInterruptStatus](#) (RDC\_Type \*base, uint32\_t mask)  
*Clear interrupt pending status.*
- static uint32\_t [RDC\\_GetStatus](#) (RDC\_Type \*base)  
*Get RDC status.*
- static void [RDC\\_ClearStatus](#) (RDC\_Type \*base, uint32\_t mask)  
*Clear RDC status.*
- static void [RDC\\_SetMasterDomainAssignment](#) (RDC\_Type \*base, [rdc\\_master\\_t](#) master, const [rdc\\_domain\\_assignment\\_t](#) \*assignment)  
*Set master domain assignment.*
- static void [RDC\\_GetDefaultMasterDomainAssignment](#) ([rdc\\_domain\\_assignment\\_t](#) \*assignment)  
*Get default master domain assignment.*
- static void [RDC\\_LockMasterDomainAssignment](#) (RDC\_Type \*base, [rdc\\_master\\_t](#) master)  
*Lock master domain assignment.*
- void [RDC\\_SetPeriphAccessConfig](#) (RDC\_Type \*base, const [rdc\\_periph\\_access\\_config\\_t](#) \*config)  
*Set peripheral access policy.*
- void [RDC\\_GetDefaultPeriphAccessConfig](#) ([rdc\\_periph\\_access\\_config\\_t](#) \*config)  
*Get default peripheral access policy.*
- static void [RDC\\_LockPeriphAccessConfig](#) (RDC\_Type \*base, [rdc\\_periph\\_t](#) periph)  
*Lock peripheral access policy configuration.*
- void [RDC\\_SetMemAccessConfig](#) (RDC\_Type \*base, const [rdc\\_mem\\_access\\_config\\_t](#) \*config)  
*Set memory region access policy.*
- void [RDC\\_GetDefaultMemAccessConfig](#) ([rdc\\_mem\\_access\\_config\\_t](#) \*config)  
*Get default memory region access policy.*
- static void [RDC\\_LockMemAccessConfig](#) (RDC\_Type \*base, [rdc\\_mem\\_t](#) mem)  
*Lock memory access policy configuration.*
- static void [RDC\\_SetMemAccessValid](#) (RDC\_Type \*base, [rdc\\_mem\\_t](#) mem, bool valid)  
*Enable or disable memory access policy configuration.*
- void [RDC\\_GetMemViolationStatus](#) (RDC\_Type \*base, [rdc\\_mem\\_t](#) mem, [rdc\\_mem\\_status\\_t](#) \*status)  
*Get the memory region violation status.*
- static void [RDC\\_ClearMemViolationFlag](#) (RDC\_Type \*base, [rdc\\_mem\\_t](#) mem)  
*Clear the memory region violation flag.*
- static uint8\_t [RDC\\_GetCurrentMasterDomainId](#) (RDC\_Type \*base)  
*Gets the domain ID of the current bus master.*

## 14.2 Data Structure Documentation

### 14.2.1 struct rdc\_hardware\_config\_t

#### Data Fields

- uint32\_t [domainNumber](#): 4  
*Number of domains.*
- uint32\_t [masterNumber](#): 8  
*Number of bus masters.*
- uint32\_t [periphNumber](#): 8  
*Number of peripherals.*
- uint32\_t [memNumber](#): 8  
*Number of memory regions.*

#### 14.2.1.0.0.12 Field Documentation

14.2.1.0.0.12.1 uint32\_t rdc\_hardware\_config\_t::domainNumber

14.2.1.0.0.12.2 uint32\_t rdc\_hardware\_config\_t::masterNumber

14.2.1.0.0.12.3 uint32\_t rdc\_hardware\_config\_t::periphNumber

14.2.1.0.0.12.4 uint32\_t rdc\_hardware\_config\_t::memNumber

### 14.2.2 struct rdc\_domain\_assignment\_t

#### Data Fields

- uint32\_t [domainId](#): 2U  
*Domain ID.*
- uint32\_t [\\_\\_pad0\\_\\_](#): 29U  
*Reserved.*
- uint32\_t [lock](#): 1U  
*Lock the domain assignment.*

#### 14.2.2.0.0.13 Field Documentation

14.2.2.0.0.13.1 uint32\_t rdc\_domain\_assignment\_t::domainId

14.2.2.0.0.13.2 uint32\_t rdc\_domain\_assignment\_t::\_\_pad0\_\_

14.2.2.0.0.13.3 uint32\_t rdc\_domain\_assignment\_t::lock

### 14.2.3 struct rdc\_periph\_access\_config\_t

#### Data Fields

- rdc\_periph\_t [periph](#)

## Data Structure Documentation

- *Peripheral name.*  
• bool [lock](#)  
*Lock the permission until reset.*
- bool [enableSema](#)  
*Enable semaphore or not, when enabled, master should call [RDC\\_SEMA42\\_Lock](#) to lock the semaphore gate accordingly before access the peripheral.*
- uint16\_t [policy](#)  
*Access policy.*

### 14.2.3.0.0.14 Field Documentation

14.2.3.0.0.14.1 `rdc_periph_t rdc_periph_access_config_t::periph`

14.2.3.0.0.14.2 `bool rdc_periph_access_config_t::lock`

14.2.3.0.0.14.3 `bool rdc_periph_access_config_t::enableSema`

14.2.3.0.0.14.4 `uint16_t rdc_periph_access_config_t::policy`

### 14.2.4 `struct rdc_mem_access_config_t`

Note that when setting the [baseAddress](#) and [endAddress](#), should be aligned to the region resolution, see `rdc_mem_t` definitions.

## Data Fields

- `rdc_mem_t` [mem](#)  
*Memory region descriptor name.*
- bool [lock](#)  
*Lock the configuration.*
- uint32\_t [baseAddress](#)  
*Start address of the memory region.*
- uint32\_t [endAddress](#)  
*End address of the memory region.*
- uint16\_t [policy](#)  
*Access policy.*

### 14.2.4.0.0.15 Field Documentation

- 14.2.4.0.0.15.1 `rdc_mem_t rdc_mem_access_config_t::mem`
- 14.2.4.0.0.15.2 `bool rdc_mem_access_config_t::lock`
- 14.2.4.0.0.15.3 `uint32_t rdc_mem_access_config_t::baseAddress`
- 14.2.4.0.0.15.4 `uint32_t rdc_mem_access_config_t::endAddress`
- 14.2.4.0.0.15.5 `uint16_t rdc_mem_access_config_t::policy`

### 14.2.5 struct `rdc_mem_status_t`

#### Data Fields

- `bool hasViolation`  
*Violating happens or not.*
- `uint8_t domainID`  
*Violating Domain ID.*
- `uint32_t address`  
*Violating Address.*

### 14.2.5.0.0.16 Field Documentation

- 14.2.5.0.0.16.1 `bool rdc_mem_status_t::hasViolation`
- 14.2.5.0.0.16.2 `uint8_t rdc_mem_status_t::domainID`
- 14.2.5.0.0.16.3 `uint32_t rdc_mem_status_t::address`

## 14.3 Macro Definition Documentation

### 14.3.1 `#define FSL_RDC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

## 14.4 Enumeration Type Documentation

### 14.4.1 enum `_rdc_interrupts`

Enumerator

***kRDC\_RestoreCompleteInterrupt*** Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions.

## Function Documentation

### 14.4.2 enum \_rdc\_flags

Enumerator

***kRDC\_PowerDownDomainOn*** Power down domain is ON.

### 14.4.3 enum \_rdc\_access\_policy

Enumerator

***kRDC\_NoAccess*** Could not read or write.

***kRDC\_WriteOnly*** Write only.

***kRDC\_ReadOnly*** Read only.

***kRDC\_ReadWrite*** Read and write.

## 14.5 Function Documentation

### 14.5.1 void RDC\_Init ( RDC\_Type \* *base* )

This function enables the RDC clock.

Parameters

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

### 14.5.2 void RDC\_Deinit ( RDC\_Type \* *base* )

This function disables the RDC clock.

Parameters

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

### 14.5.3 static void RDC\_GetHardwareConfig ( RDC\_Type \* *base*, rdc\_hardware\_config\_t \* *config* ) [inline], [static]

This function gets the RDC hardware configurations, including number of bus masters, number of domains, number of memory regions and number of peripherals.

## Parameters

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

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

## Parameters

<i>base</i>	RDC peripheral base address.
<i>mask</i>	Interrupts to enable, it is OR'ed value of enum <a href="#">_rdc_interrupts</a> .

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

## Parameters

<i>base</i>	RDC peripheral base address.
<i>mask</i>	Interrupts to disable, it is OR'ed value of enum <a href="#">_rdc_interrupts</a> .

#### 14.5.6 static uint32\_t RDC\_GetInterruptStatus ( RDC\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

Interrupts pending status, it is OR'ed value of enum [\\_rdc\\_interrupts](#).

#### 14.5.7 static void RDC\_ClearInterruptStatus ( RDC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

### Parameters

<i>base</i>	RDC peripheral base address.
<i>mask</i>	Status to clear, it is OR'ed value of enum <a href="#">_rdc_interrupts</a> .

### 14.5.8 static uint32\_t RDC\_GetStatus ( RDC\_Type \* *base* ) [inline], [static]

### Parameters

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

### Returns

mask RDC status, it is OR'ed value of enum [\\_rdc\\_flags](#).

### 14.5.9 static void RDC\_ClearStatus ( RDC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

<i>base</i>	RDC peripheral base address.
<i>mask</i>	RDC status to clear, it is OR'ed value of enum <a href="#">_rdc_flags</a> .

### 14.5.10 static void RDC\_SetMasterDomainAssignment ( RDC\_Type \* *base*, rdc\_master\_t *master*, const rdc\_domain\_assignment\_t \* *assignment* ) [inline], [static]

### Parameters

<i>base</i>	RDC peripheral base address.
<i>master</i>	Which master to set.



<i>assignment</i>	Pointer to the assignment.
-------------------	----------------------------

#### 14.5.11 static void RDC\_GetDefaultMasterDomainAssignment ( rdc\_domain\_assignment\_t \* *assignment* ) [inline], [static]

The default configuration is:

```
assignment->domainId = 0U;
assignment->lock = 0U;
```

Parameters

<i>assignment</i>	Pointer to the assignment.
-------------------	----------------------------

#### 14.5.12 static void RDC\_LockMasterDomainAssignment ( RDC\_Type \* *base*, rdc\_master\_t *master* ) [inline], [static]

Once locked, it could not be unlocked until next reset.

Parameters

<i>base</i>	RDC peripheral base address.
<i>master</i>	Which master to lock.

#### 14.5.13 void RDC\_SetPeriphAccessConfig ( RDC\_Type \* *base*, const rdc\_periph\_access\_config\_t \* *config* )

Parameters

<i>base</i>	RDC peripheral base address.
<i>config</i>	Pointer to the policy configuration.

#### 14.5.14 void RDC\_GetDefaultPeriphAccessConfig ( rdc\_periph\_access\_config\_t \* *config* )

The default configuration is:

## Function Documentation

```
config->lock = false;
config->enableSema = false;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

### Parameters

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

### 14.5.15 static void RDC\_LockPeriphAccessConfig ( RDC\_Type \* *base*, rdc\_periph\_t *periph* ) [inline], [static]

Once locked, it could not be unlocked until reset.

### Parameters

<i>base</i>	RDC peripheral base address.
<i>periph</i>	Which peripheral to lock.

### 14.5.16 void RDC\_SetMemAccessConfig ( RDC\_Type \* *base*, const rdc\_mem\_access\_config\_t \* *config* )

Note that when setting the baseAddress and endAddress in *config*, should be aligned to the region resolution, see *rdc\_mem\_t* definitions.

### Parameters

<i>base</i>	RDC peripheral base address.
<i>config</i>	Pointer to the policy configuration.

### 14.5.17 void RDC\_GetDefaultMemAccessConfig ( rdc\_mem\_access\_config\_t \* *config* )

The default configuration is:

```
config->lock = false;
config->baseAddress = 0;
config->endAddress = 0;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
                RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

## Parameters

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

#### 14.5.18 static void RDC\_LockMemAccessConfig ( RDC\_Type \* *base*, rdc\_mem\_t *mem* ) [inline], [static]

Once locked, it could not be unlocked until reset. After locked, you can only call [RDC\\_SetMemAccessValid](#) to enable the configuration, but can not disable it or change other settings.

## Parameters

<i>base</i>	RDC peripheral base address.
<i>mem</i>	Which memory region to lock.

#### 14.5.19 static void RDC\_SetMemAccessValid ( RDC\_Type \* *base*, rdc\_mem\_t *mem*, bool *valid* ) [inline], [static]

## Parameters

<i>base</i>	RDC peripheral base address.
<i>mem</i>	Which memory region to operate.
<i>valid</i>	Pass in true to valid, false to invalid.

#### 14.5.20 void RDC\_GetMemViolationStatus ( RDC\_Type \* *base*, rdc\_mem\_t *mem*, rdc\_mem\_status\_t \* *status* )

The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID(s).

## Parameters

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

## Function Documentation

<i>mem</i>	Which memory region to get.
<i>status</i>	The returned status.

**14.5.21 static void RDC\_ClearMemViolationFlag ( RDC\_Type \* *base*, rdc\_mem\_t *mem* ) [inline], [static]**

Parameters

<i>base</i>	RDC peripheral base address.
<i>mem</i>	Which memory region to clear.

**14.5.22 static uint8\_t RDC\_GetCurrentMasterDomainId ( RDC\_Type \* *base* ) [inline], [static]**

This function returns the domain ID of the current bus master.

Parameters

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

Returns

Domain ID of current bus master.

## Chapter 15

# RDC\_SEMA42: Hardware Semaphores Driver

### 15.1 Overview

The MCUXpresso SDK provides a driver for the RDC\_SEMA42 module of MCUXpresso SDK devices.

The RDC\_SEMA42 driver should be used together with RDC driver.

Before using the RDC\_SEMA42, call the [RDC\\_SEMA42\\_Init\(\)](#) function to initialize the module. Note that this function only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either the [RDC\\_SEMA42\\_ResetGate\(\)](#) or [RDC\\_SEMA42\\_ResetAllGates\(\)](#) functions. The function [RDC\\_SEMA42\\_Deinit\(\)](#) deinitializes the RDC\_SEMA42.

The RDC\_SEMA42 provides two functions to lock the RDC\_SEMA42 gate. The function [RDC\\_SEMA42\\_TryLock\(\)](#) tries to lock the gate. If the gate has been locked by another processor, this function returns an error immediately. The function [RDC\\_SEMA42\\_Lock\(\)](#) is a blocking method, which waits until the gate is free and locks it.

The [RDC\\_SEMA42\\_Unlock\(\)](#) unlocks the RDC\_SEMA42 gate. The gate can only be unlocked by the processor which locked it. If the gate is not locked by the current processor, this function takes no effect. The function [RDC\\_SEMA42\\_GetGateStatus\(\)](#) returns a status whether the gate is unlocked and which processor locks the gate. The function [RDC\\_SEMA42\\_GetLockDomainID\(\)](#) returns the ID of the domain which has locked the gate.

The RDC\_SEMA42 gate can be reset to unlock forcefully. The function [RDC\\_SEMA42\\_ResetGate\(\)](#) resets a specific gate. The function [RDC\\_SEMA42\\_ResetAllGates\(\)](#) resets all gates.

### Macros

- #define [RDC\\_SEMA42\\_GATE\\_NUM\\_RESET\\_ALL](#) (64U)  
*The number to reset all RDC\_SEMA42 gates.*
- #define [RDC\\_SEMA42\\_GATEn](#)(base, n) (\*( &((base)->GATE0) + (n)))  
*RDC\_SEMA42 gate n register address.*
- #define [RDC\\_SEMA42\\_GATE\\_COUNT](#) (64U)  
*RDC\_SEMA42 gate count.*

### Functions

- void [RDC\\_SEMA42\\_Init](#) (RDC\_SEMAPHORE\_Type \*base)  
*Initializes the RDC\_SEMA42 module.*
- void [RDC\\_SEMA42\\_Deinit](#) (RDC\_SEMAPHORE\_Type \*base)  
*De-initializes the RDC\_SEMA42 module.*
- status\_t [RDC\\_SEMA42\\_TryLock](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum, uint8\_t masterIndex, uint8\_t domainId)  
*Tries to lock the RDC\_SEMA42 gate.*

## Function Documentation

- void [RDC\\_SEMA42\\_Lock](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum, uint8\_t masterIndex, uint8\_t domainId)  
*Locks the RDC\_SEMA42 gate.*
- static void [RDC\\_SEMA42\\_Unlock](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Unlocks the RDC\_SEMA42 gate.*
- static int32\_t [RDC\\_SEMA42\\_GetLockMasterIndex](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Gets which master has currently locked the gate.*
- int32\_t [RDC\\_SEMA42\\_GetLockDomainID](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Gets which domain has currently locked the gate.*
- status\_t [RDC\\_SEMA42\\_ResetGate](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Resets the RDC\_SEMA42 gate to an unlocked status.*
- static status\_t [RDC\\_SEMA42\\_ResetAllGates](#) (RDC\_SEMAPHORE\_Type \*base)  
*Resets all RDC\_SEMA42 gates to an unlocked status.*

## Driver version

- #define [FSL\\_RDC\\_SEMA42\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*RDC\_SEMA42 driver version.*

## 15.2 Macro Definition Documentation

### 15.2.1 #define RDC\_SEMA42\_GATE\_NUM\_RESET\_ALL (64U)

### 15.2.2 #define RDC\_SEMA42\_GATEn( base, n ) (\*( &((base)->GATE0) + (n)))

### 15.2.3 #define RDC\_SEMA42\_GATE\_COUNT (64U)

## 15.3 Function Documentation

### 15.3.1 void RDC\_SEMA42\_Init ( RDC\_SEMAPHORE\_Type \* base )

This function initializes the RDC\_SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either RDC\_SEMA42\_ResetGate or RDC\_SEMA42\_ResetAllGates function.

Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
-------------	-------------------------------------

### 15.3.2 void RDC\_SEMA42\_Deinit ( RDC\_SEMAPHORE\_Type \* base )

This function de-initializes the RDC\_SEMA42 module. It only disables the clock.

## Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
-------------	-------------------------------------

### 15.3.3 **status\_t RDC\_SEMA42\_TryLock ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *masterIndex*, uint8\_t *domainId* )**

This function tries to lock the specific RDC\_SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

## Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
<i>gateNum</i>	Gate number to lock.
<i>masterIndex</i>	Current processor master index.
<i>domainId</i>	Current processor domain ID.

## Return values

<i>kStatus_Success</i>	Lock the sema42 gate successfully.
<i>kStatus_Failed</i>	Sema42 gate has been locked by another processor.

### 15.3.4 **void RDC\_SEMA42\_Lock ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *masterIndex*, uint8\_t *domainId* )**

This function locks the specific RDC\_SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

## Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
<i>gateNum</i>	Gate number to lock.
<i>masterIndex</i>	Current processor master index.
<i>domainId</i>	Current processor domain ID.

## Function Documentation

### 15.3.5 static void RDC\_SEMA42\_Unlock ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function unlocks the specific RDC\_SEMA42 gate. It only writes unlock value to the RDC\_SEMA42 gate register. However, it does not check whether the RDC\_SEMA42 gate is locked by the current processor or not. As a result, if the RDC\_SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
<i>gateNum</i>	Gate number to unlock.

### 15.3.6 static int32\_t RDC\_SEMA42\_GetLockMasterIndex ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
<i>gateNum</i>	Gate number.

Returns

Return -1 if the gate is not locked by any master, otherwise return the master index.

### 15.3.7 int32\_t RDC\_SEMA42\_GetLockDomainID ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum* )

Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
<i>gateNum</i>	Gate number.

Returns

Return -1 if the gate is not locked by any domain, otherwise return the domain ID.



### 15.3.8 `status_t RDC_SEMA42_ResetGate ( RDC_SEMAPHORE_Type * base, uint8_t gateNum )`

This function resets a RDC\_SEMA42 gate to an unlocked status.

## Function Documentation

### Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
<i>gateNum</i>	Gate number.

### Return values

<i>kStatus_Success</i>	RDC_SEMA42 gate is reset successfully.
<i>kStatus_Failed</i>	Some other reset process is ongoing.

### 15.3.9 static status\_t RDC\_SEMA42\_ResetAllGates ( RDC\_SEMAPHORE\_Type \* *base* ) [inline], [static]

This function resets all RDC\_SEMA42 gate to an unlocked status.

### Parameters

<i>base</i>	RDC_SEMA42 peripheral base address.
-------------	-------------------------------------

### Return values

<i>kStatus_Success</i>	RDC_SEMA42 is reset successfully.
<i>kStatus_RDC_SEMA42_-Reseting</i>	Some other reset process is ongoing.

## Chapter 16

### SAI: Serial Audio Interface

#### 16.1 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.

#### 16.2 Typical use case

##### 16.2.1 SAI Send/receive using an interrupt method

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

##### 16.2.2 SAI Send/receive using a DMA method

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

#### Modules

- [SAI DMA Driver](#)
- [SAI EDMA Driver](#)
- [SAI SDMA Driver](#)

## Typical use case

## Data Structures

- struct `sai_config_t`  
*SAI user configuration structure. [More...](#)*
- struct `sai_transfer_format_t`  
*sai transfer format [More...](#)*
- struct `sai_transfer_t`  
*SAI transfer structure. [More...](#)*
- struct `sai_handle_t`  
*SAI handle structure. [More...](#)*

## Macros

- #define `SAI_XFER_QUEUE_SIZE` (4)  
*SAI transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* `sai_transfer_callback_t`)(I2S\_Type \*base, sai\_handle\_t \*handle, status\_t status, void \*userData)  
*SAI transfer callback prototype.*

## Enumerations

- enum `_sai_status_t` {  
    `kStatus_SAI_TxBusy` = MAKE\_STATUS(kStatusGroup\_SAI, 0),  
    `kStatus_SAI_RxBusy` = MAKE\_STATUS(kStatusGroup\_SAI, 1),  
    `kStatus_SAI_TxError` = MAKE\_STATUS(kStatusGroup\_SAI, 2),  
    `kStatus_SAI_RxError` = MAKE\_STATUS(kStatusGroup\_SAI, 3),  
    `kStatus_SAI_QueueFull` = MAKE\_STATUS(kStatusGroup\_SAI, 4),  
    `kStatus_SAI_TxIdle` = MAKE\_STATUS(kStatusGroup\_SAI, 5),  
    `kStatus_SAI_RxIdle` = MAKE\_STATUS(kStatusGroup\_SAI, 6) }  
*SAI return status.*
- enum `_sai_channel_mask` {  
    `kSAI_Channel0Mask` = 1 << 0U,  
    `kSAI_Channel1Mask` = 1 << 1U,  
    `kSAI_Channel2Mask` = 1 << 2U,  
    `kSAI_Channel3Mask` = 1 << 3U,  
    `kSAI_Channel4Mask` = 1 << 4U,  
    `kSAI_Channel5Mask` = 1 << 5U,  
    `kSAI_Channel6Mask` = 1 << 6U,  
    `kSAI_Channel7Mask` = 1 << 7U }  
• enum `sai_protocol_t` {  
    `kSAI_BusLeftJustified` = 0x0U,  
    `kSAI_BusRightJustified`,  
    `kSAI_BusI2S`,  
    `kSAI_BusPCMA`,  
    `kSAI_BusPCMB` }

- Define the SAI bus type.*

  - enum `sai_master_slave_t` {  
`kSAI_Master` = 0x0U,  
`kSAI_Slave` = 0x1U }

*Master or slave mode.*
- enum `sai_mono_stereo_t` {  
`kSAI_Stereo` = 0x0U,  
`kSAI_MonoRight`,  
`kSAI_MonoLeft` }

*Mono or stereo audio format.*
- enum `sai_data_order_t` {  
`kSAI_DataLSB` = 0x0U,  
`kSAI_DataMSB` }

*SAI data order, MSB or LSB.*
- enum `sai_clock_polarity_t` {  
`kSAI_PolarityActiveHigh` = 0x0U,  
`kSAI_PolarityActiveLow` }

*SAI clock polarity, active high or low.*
- enum `sai_sync_mode_t` {  
`kSAI_ModeAsync` = 0x0U,  
`kSAI_ModeSync`,  
`kSAI_ModeSyncWithOtherTx`,  
`kSAI_ModeSyncWithOtherRx` }

*Synchronous or asynchronous mode.*
- enum `sai_mclk_source_t` {  
`kSAI_MclkSourceSysclk` = 0x0U,  
`kSAI_MclkSourceSelect1`,  
`kSAI_MclkSourceSelect2`,  
`kSAI_MclkSourceSelect3` }

*Master clock source.*
- enum `sai_bclk_source_t` {  
`kSAI_BclkSourceBusclk` = 0x0U,  
`kSAI_BclkSourceMclkOption1` = 0x1U,  
`kSAI_BclkSourceMclkOption2` = 0x2U,  
`kSAI_BclkSourceMclkOption3` = 0x3U,  
`kSAI_BclkSourceMclkDiv` = 0x1U,  
`kSAI_BclkSourceOtherSai0` = 0x2U,  
`kSAI_BclkSourceOtherSai1` = 0x3U }

*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` {

## Typical use case

```
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_fifo_packing_t` {  
    kSAI\_FifoPackingDisabled = 0x0U,  
    kSAI\_FifoPacking8bit = 0x2U,  
    kSAI\_FifoPacking16bit = 0x3U }

*The SAI packing mode The mode includes 8 bit and 16 bit packing.*

- enum `sai_sample_rate_t` {  
    kSAI\_SampleRate8KHz = 8000U,  
    kSAI\_SampleRate11025Hz = 11025U,  
    kSAI\_SampleRate12KHz = 12000U,  
    kSAI\_SampleRate16KHz = 16000U,  
    kSAI\_SampleRate22050Hz = 22050U,  
    kSAI\_SampleRate24KHz = 24000U,  
    kSAI\_SampleRate32KHz = 32000U,  
    kSAI\_SampleRate44100Hz = 44100U,  
    kSAI\_SampleRate48KHz = 48000U,  
    kSAI\_SampleRate96KHz = 96000U,  
    kSAI\_SampleRate192KHz = 192000U,  
    kSAI\_SampleRate384KHz = 384000U }

*Audio sample rate.*

- enum `sai_word_width_t` {  
    kSAI\_WordWidth8bits = 8U,  
    kSAI\_WordWidth16bits = 16U,  
    kSAI\_WordWidth24bits = 24U,  
    kSAI\_WordWidth32bits = 32U }

*Audio word width.*

## Driver version

- #define `FSL_SAI_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 7))  
    *Version 2.1.7.*

## Initialization and deinitialization

- void [SAI\\_TxInit](#) (I2S\_Type \*base, const [sai\\_config\\_t](#) \*config)  
*Initializes the SAI Tx peripheral.*
- void [SAI\\_RxInit](#) (I2S\_Type \*base, const [sai\\_config\\_t](#) \*config)  
*Initializes the SAI Rx peripheral.*
- void [SAI\\_TxGetDefaultConfig](#) ([sai\\_config\\_t](#) \*config)  
*Sets the SAI Tx configuration structure to default values.*
- void [SAI\\_RxGetDefaultConfig](#) ([sai\\_config\\_t](#) \*config)  
*Sets the SAI Rx configuration structure to default values.*
- void [SAI\\_Deinit](#) (I2S\_Type \*base)  
*De-initializes the SAI peripheral.*
- void [SAI\\_TxReset](#) (I2S\_Type \*base)  
*Resets the SAI Tx.*
- void [SAI\\_RxReset](#) (I2S\_Type \*base)  
*Resets the SAI Rx.*
- void [SAI\\_TxEnable](#) (I2S\_Type \*base, bool enable)  
*Enables/disables the SAI Tx.*
- void [SAI\\_RxEnable](#) (I2S\_Type \*base, bool enable)  
*Enables/disables the SAI Rx.*

## Status

- static uint32\_t [SAI\\_TxGetStatusFlag](#) (I2S\_Type \*base)  
*Gets the SAI Tx status flag state.*
- static void [SAI\\_TxClearStatusFlags](#) (I2S\_Type \*base, uint32\_t mask)  
*Clears the SAI Tx status flag state.*
- static uint32\_t [SAI\\_RxGetStatusFlag](#) (I2S\_Type \*base)  
*Gets the SAI Rx status flag state.*
- static void [SAI\\_RxClearStatusFlags](#) (I2S\_Type \*base, uint32\_t mask)  
*Clears the SAI Rx status flag state.*
- void [SAI\\_TxSoftwareReset](#) (I2S\_Type \*base, [sai\\_reset\\_type\\_t](#) type)  
*Do software reset or FIFO reset .*
- void [SAI\\_RxSoftwareReset](#) (I2S\_Type \*base, [sai\\_reset\\_type\\_t](#) type)  
*Do software reset or FIFO reset .*
- void [SAI\\_TxSetChannelFIFOMask](#) (I2S\_Type \*base, uint8\_t mask)  
*Set the Tx channel FIFO enable mask.*
- void [SAI\\_RxSetChannelFIFOMask](#) (I2S\_Type \*base, uint8\_t mask)  
*Set the Rx channel FIFO enable mask.*
- void [SAI\\_TxSetDataOrder](#) (I2S\_Type \*base, [sai\\_data\\_order\\_t](#) order)  
*Set the Tx data order.*
- void [SAI\\_RxSetDataOrder](#) (I2S\_Type \*base, [sai\\_data\\_order\\_t](#) order)  
*Set the Rx data order.*
- void [SAI\\_TxSetBitClockPolarity](#) (I2S\_Type \*base, [sai\\_clock\\_polarity\\_t](#) polarity)  
*Set the Tx data order.*
- void [SAI\\_RxSetBitClockPolarity](#) (I2S\_Type \*base, [sai\\_clock\\_polarity\\_t](#) polarity)  
*Set the Rx data order.*
- void [SAI\\_TxSetFrameSyncPolarity](#) (I2S\_Type \*base, [sai\\_clock\\_polarity\\_t](#) polarity)  
*Set the Tx data order.*
- void [SAI\\_RxSetFrameSyncPolarity](#) (I2S\_Type \*base, [sai\\_clock\\_polarity\\_t](#) polarity)  
*Set the Rx data order.*
- void [SAI\\_TxSetFIFOPacking](#) (I2S\_Type \*base, [sai\\_fifo\\_packing\\_t](#) pack)

## Typical use case

- *Set Tx FIFO packing feature.*  
void [SAI\\_RxSetFIFOPacking](#) (I2S\_Type \*base, [sai\\_fifo\\_packing\\_t](#) pack)
- *Set Rx FIFO packing feature.*  
static void [SAI\\_TxSetFIFOErrorContinue](#) (I2S\_Type \*base, bool isEnabled)
- *Set Tx FIFO error continue.*  
static void [SAI\\_RxSetFIFOErrorContinue](#) (I2S\_Type \*base, bool isEnabled)
- *Set Rx FIFO error continue.*

## Interrupts

- static void [SAI\\_TxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables the SAI Tx interrupt requests.*
- static void [SAI\\_RxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables the SAI Rx interrupt requests.*
- static void [SAI\\_TxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables the SAI Tx interrupt requests.*
- static void [SAI\\_RxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables the SAI Rx interrupt requests.*

## DMA Control

- static void [SAI\\_TxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the SAI Tx DMA requests.*
- static void [SAI\\_RxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the SAI Rx DMA requests.*
- static uint32\_t [SAI\\_TxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Tx data register address.*
- static uint32\_t [SAI\\_RxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Rx data register address.*

## Bus Operations

- void [SAI\\_TxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_RxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- void [SAI\\_WriteBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Sends data using a blocking method.*
- void [SAI\\_WriteMultiChannelBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t channelMask, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Sends data to multi channel using a blocking method.*
- static void [SAI\\_WriteData](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t data)  
*Writes data into SAI FIFO.*
- void [SAI\\_ReadBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Receives data using a blocking method.*



- void [SAI\\_ReadMultiChannelBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t channelMask, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Receives multi channel data using a blocking method.*
- static uint32\_t [SAI\\_ReadData](#) (I2S\_Type \*base, uint32\_t channel)  
*Reads data from the SAI FIFO.*

## Transactional

- void [SAI\\_TransferTxCreateHandle](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_callback\_t callback, void \*userData)  
*Initializes the SAI Tx handle.*
- void [SAI\\_TransferRxCreateHandle](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_callback\_t callback, void \*userData)  
*Initializes the SAI Rx handle.*
- status\_t [SAI\\_TransferTxSetFormat](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- status\_t [SAI\\_TransferRxSetFormat](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- status\_t [SAI\\_TransferSendNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking send transfer on SAI.*
- status\_t [SAI\\_TransferReceiveNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking receive transfer on SAI.*
- status\_t [SAI\\_TransferGetSendCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a set byte count.*
- status\_t [SAI\\_TransferGetReceiveCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a received byte count.*
- void [SAI\\_TransferAbortSend](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Aborts the current send.*
- void [SAI\\_TransferAbortReceive](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Aborts the current IRQ receive.*
- void [SAI\\_TransferTerminateSend](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Terminate all SAI send.*
- void [SAI\\_TransferTerminateReceive](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Terminate all SAI receive.*
- void [SAI\\_TransferTxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Tx interrupt handler.*
- void [SAI\\_TransferRxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Tx interrupt handler.*

### 16.3 Data Structure Documentation

#### 16.3.1 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\\_bclk\\_source\\_t bclkSource](#)  
*Bit Clock source.*
- [sai\\_master\\_slave\\_t masterSlave](#)  
*Master or slave.*

#### 16.3.2 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](#)  
*Transfer start channel.*
- [uint8\\_t channelMask](#)  
*enabled channel mask value, reference `_sai_channel_mask`*
- [uint8\\_t endChannel](#)  
*end channel number*
- [uint8\\_t channelNums](#)  
*Total enabled channel numbers.*
- [sai\\_protocol\\_t protocol](#)  
*Which audio protocol used.*
- [bool isFrameSyncCompact](#)  
*True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.*

**16.3.2.0.0.17 Field Documentation****16.3.2.0.0.17.1 bool sai\_transfer\_format\_t::isFrameSyncCompact****16.3.3 struct sai\_transfer\_t****Data Fields**

- uint8\_t \* [data](#)  
*Data start address to transfer.*
- size\_t [dataSize](#)  
*Transfer size.*

**16.3.3.0.0.18 Field Documentation****16.3.3.0.0.18.1 uint8\_t\* sai\_transfer\_t::data****16.3.3.0.0.18.2 size\_t sai\_transfer\_t::dataSize****16.3.4 struct \_sai\_handle****Data Fields**

- I2S\_Type \* [base](#)  
*base address*
- uint32\_t [state](#)  
*Transfer status.*
- [sai\\_transfer\\_callback\\_t](#) [callback](#)  
*Callback function called at transfer event.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*
- uint8\_t [bitWidth](#)  
*Bit width for transfer, 8/16/24/32 bits.*
- uint8\_t [channel](#)  
*Transfer start channel.*
- uint8\_t [channelMask](#)  
*enabled channel mask value, refernece \_sai\_channel\_mask*
- uint8\_t [endChannel](#)  
*end channel number*
- uint8\_t [channelNums](#)  
*Total enabled channel numbers.*
- [sai\\_transfer\\_t](#) [saiQueue](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*

## Enumeration Type Documentation

- uint8\_t [watermark](#)  
Watermark value.

## 16.4 Macro Definition Documentation

### 16.4.1 #define SAI\_XFER\_QUEUE\_SIZE (4)

## 16.5 Enumeration Type Documentation

### 16.5.1 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.

### 16.5.2 enum \_sai\_channel\_mask

Enumerator

*kSAI\_Channel0Mask* channel 0 mask value  
*kSAI\_Channel1Mask* channel 1 mask value  
*kSAI\_Channel2Mask* channel 2 mask value  
*kSAI\_Channel3Mask* channel 3 mask value  
*kSAI\_Channel4Mask* channel 4 mask value  
*kSAI\_Channel5Mask* channel 5 mask value  
*kSAI\_Channel6Mask* channel 6 mask value  
*kSAI\_Channel7Mask* channel 7 mask value

### 16.5.3 enum sai\_protocol\_t

Enumerator

*kSAI\_BusLeftJustified* Uses left justified format.  
*kSAI\_BusRightJustified* Uses right justified format.  
*kSAI\_BusI2S* Uses I2S format.  
*kSAI\_BusPCMA* Uses I2S PCM A format.  
*kSAI\_BusPCMB* Uses I2S PCM B format.

### 16.5.4 enum sai\_master\_slave\_t

Enumerator

*kSAI\_Master* Master mode.

*kSAI\_Slave* Slave mode.

### 16.5.5 enum sai\_mono\_stereo\_t

Enumerator

*kSAI\_Stereo* Stereo sound.

*kSAI\_MonoRight* Only Right channel have sound.

*kSAI\_MonoLeft* Only left channel have sound.

### 16.5.6 enum sai\_data\_order\_t

Enumerator

*kSAI\_DataLSB* LSB bit transferred first.

*kSAI\_DataMSB* MSB bit transferred first.

### 16.5.7 enum sai\_clock\_polarity\_t

Enumerator

*kSAI\_PolarityActiveHigh* Clock active high.

*kSAI\_PolarityActiveLow* Clock active low.

### 16.5.8 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.

### 16.5.9 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.

### 16.5.10 enum sai\_bclk\_source\_t

Enumerator

*kSAI\_BclkSourceBusclk* Bit clock using bus clock.  
*kSAI\_BclkSourceMclkOption1* Bit clock MCLK option 1.  
*kSAI\_BclkSourceMclkOption2* Bit clock MCLK option2.  
*kSAI\_BclkSourceMclkOption3* Bit clock MCLK option3.  
*kSAI\_BclkSourceMclkDiv* Bit clock using master clock divider.  
*kSAI\_BclkSourceOtherSai0* Bit clock from other SAI device.  
*kSAI\_BclkSourceOtherSai1* Bit clock from other SAI device.

### 16.5.11 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.

### 16.5.12 enum \_sai\_dma\_enable\_t

Enumerator

*kSAI\_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.  
*kSAI\_FIFORequestDMAEnable* FIFO request caused by the DMA request.

### 16.5.13 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.

### 16.5.14 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.

### 16.5.15 enum sai\_fifo\_packing\_t

Enumerator

*kSAI\_FifoPackingDisabled* Packing disabled.  
*kSAI\_FifoPacking8bit* 8 bit packing enabled  
*kSAI\_FifoPacking16bit* 16bit packing enabled

### 16.5.16 enum sai\_sample\_rate\_t

Enumerator

*kSAI\_SampleRate8KHz* Sample rate 8000 Hz.  
*kSAI\_SampleRate11025Hz* Sample rate 11025 Hz.  
*kSAI\_SampleRate12KHz* Sample rate 12000 Hz.  
*kSAI\_SampleRate16KHz* Sample rate 16000 Hz.  
*kSAI\_SampleRate22050Hz* Sample rate 22050 Hz.  
*kSAI\_SampleRate24KHz* Sample rate 24000 Hz.  
*kSAI\_SampleRate32KHz* Sample rate 32000 Hz.  
*kSAI\_SampleRate44100Hz* Sample rate 44100 Hz.  
*kSAI\_SampleRate48KHz* Sample rate 48000 Hz.  
*kSAI\_SampleRate96KHz* Sample rate 96000 Hz.  
*kSAI\_SampleRate192KHz* Sample rate 192000 Hz.  
*kSAI\_SampleRate384KHz* Sample rate 384000 Hz.

## Function Documentation

### 16.5.17 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.

## 16.6 Function Documentation

### 16.6.1 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.

### 16.6.2 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.

### 16.6.3 void SAI\_TxGetDefaultConfig ( sai\_config\_t \* *config* )

This API initializes the configuration structure for use in SAI\_TxConfig(). The initialized structure can remain unchanged in SAI\_TxConfig(), or it can be modified before calling SAI\_TxConfig(). This is an example.

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

Parameters

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

### 16.6.4 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
---------------	---

### 16.6.5 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

---

## Function Documentation

<i>base</i>	SAI base pointer
-------------	------------------

### 16.6.6 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
-------------	------------------

### 16.6.7 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
-------------	------------------

### 16.6.8 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.

### 16.6.9 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.

### 16.6.10 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.

**16.6.11 static void SAI\_TxClearStatusFlags ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	State mask. It can be a combination of the following source if defined: <ul style="list-style-type: none"> <li>• kSAI_WordStartFlag</li> <li>• kSAI_SyncErrorFlag</li> <li>• kSAI_FIFOErrorFlag</li> </ul>

**16.6.12 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.

**16.6.13 static void SAI\_RxClearStatusFlags ( 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 sources if defined. <ul style="list-style-type: none"><li>• kSAI_WordStartFlag</li><li>• kSAI_SyncErrorFlag</li><li>• kSAI_FIFOErrorFlag</li></ul>

### 16.6.14 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

### 16.6.15 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

### 16.6.16 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.

**16.6.17 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.

**16.6.18 void SAI\_TxSetDataOrder ( I2S\_Type \* *base*, sai\_data\_order\_t *order* )**

## Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

**16.6.19 void SAI\_RxSetDataOrder ( I2S\_Type \* *base*, sai\_data\_order\_t *order* )**

## Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

**16.6.20 void SAI\_TxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

## Function Documentation

### Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

### 16.6.21 void SAI\_RxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

### Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

### 16.6.22 void SAI\_TxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

### Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

### 16.6.23 void SAI\_RxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

### Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

### 16.6.24 void SAI\_TxSetFIFOPacking ( I2S\_Type \* *base*, sai\_fifo\_packing\_t *pack* )

## Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

**16.6.25 void SAI\_RxSetFIFOPacking ( I2S\_Type \* *base*, sai\_fifo\_packing\_t *pack* )**

## Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

**16.6.26 static void SAI\_TxSetFIFOErrorContinue ( I2S\_Type \* *base*, bool *isEnabled* ) [inline], [static]**

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

## Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

**16.6.27 static void SAI\_RxSetFIFOErrorContinue ( I2S\_Type \* *base*, bool *isEnabled* ) [inline], [static]**

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

## Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

**16.6.28 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>

### 16.6.29 static void SAI\_RxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFOREquestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul>



### 16.6.30 static void SAI\_TxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul>

### 16.6.31 static void SAI\_RxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul>

### 16.6.32 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_FIFOResultDMAEnable</li></ul>
<i>enable</i>	True means enable DMA, false means disable DMA.

**16.6.33** `static void SAI_RxEnableDMA ( I2S_Type * base, uint32_t mask, bool enable ) [inline], [static]`

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	DMA source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"><li>• kSAI_FIFOWarningDMAEnable</li><li>• kSAI_FIFOResultDMAEnable</li></ul>
<i>enable</i>	True means enable DMA, false means disable DMA.

**16.6.34** `static uint32_t SAI_TxGetDataRegisterAddress ( I2S_Type * base, uint32_t channel ) [inline], [static]`

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

**16.6.35** `static uint32_t SAI_RxGetDataRegisterAddress ( I2S_Type * base, uint32_t channel ) [inline], [static]`

This API is used to provide a transfer address for the SAI DMA transfer configuration.

## Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

## Returns

data register address.

**16.6.36 void SAI\_TxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

<i>base</i>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

**16.6.37 void SAI\_RxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

<i>base</i>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

## Function Documentation

**16.6.38 void SAI\_WriteBlocking ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *bitWidth*, uint8\_t \* *buffer*, uint32\_t *size* )**

### Note

This function blocks by polling until data is ready to be sent.

### Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

**16.6.39 void SAI\_WriteMultiChannelBlocking ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *channelMask*, uint32\_t *bitWidth*, uint8\_t \* *buffer*, uint32\_t *size* )**

### Note

This function blocks by polling until data is ready to be sent.

### Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

**16.6.40 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.

**16.6.41 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.

**16.6.42 void SAI\_ReadMultiChannelBlocking ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *channelMask*, uint32\_t *bitWidth*, uint8\_t \* *buffer*, uint32\_t *size* )**

## Note

This function blocks by polling until data is ready to be sent.

## Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

**16.6.43**   `static uint32_t SAI_ReadData ( I2S_Type * base, uint32_t channel )`  
          `[inline], [static]`

## Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.

## Returns

Data in SAI FIFO.

#### 16.6.44 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

#### 16.6.45 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.

**16.6.46** `status_t SAI_TransferTxSetFormat ( I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.



## Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<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.

**16.6.47 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.

**16.6.48 status\_t SAI\_TransferSendNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Function Documentation

### Note

This API returns immediately after the transfer initiates. Call the SAI\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

### Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the sai_transfer_t structure.

### Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_TxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

## 16.6.49 status\_t SAI\_TransferReceiveNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

### Note

This API returns immediately after the transfer initiates. Call the SAI\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

### Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the sai_transfer_t structure.

### Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

**16.6.50** `status_t SAI_TransferGetSendCount ( I2S_Type * base, sai_handle_t * handle, size_t * count )`

## Function Documentation

### Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count sent.

### Return values

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

### 16.6.51 status\_t SAI\_TransferGetReceiveCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count received.

### Return values

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

### 16.6.52 void SAI\_TransferAbortSend ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

### Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

## Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.

**16.6.53 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.

**16.6.54 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.

**16.6.55 void SAI\_TransferTerminateReceive ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )**

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortReceive.

## Parameters

## Function Documentation

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

### 16.6.56 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.

### 16.6.57 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.

## **16.7 SAI DMA Driver**

## **16.8 SAI EDMA Driver**



## **16.9 SAI SDMA Driver**



## Chapter 17

# SEMA4: Hardware Semaphores Driver

### 17.1 Overview

The MCUXpresso SDK provides a driver for the SEMA4 module of MCUXpresso SDK devices.

#### Macros

- #define [SEMA4\\_GATE\\_NUM\\_RESET\\_ALL](#) (64U)  
*The number to reset all SEMA4 gates.*
- #define [SEMA4\\_GATEn](#)(base, n) (&((base)->Gate00) + (n))  
*SEMA4 gate n register address.*

#### Functions

- void [SEMA4\\_Init](#) (SEMA4\_Type \*base)  
*Initializes the SEMA4 module.*
- void [SEMA4\\_Deinit](#) (SEMA4\_Type \*base)  
*De-initializes the SEMA4 module.*
- status\_t [SEMA4\\_TryLock](#) (SEMA4\_Type \*base, uint8\_t gateNum, uint8\_t procNum)  
*Tries to lock the SEMA4 gate.*
- void [SEMA4\\_Lock](#) (SEMA4\_Type \*base, uint8\_t gateNum, uint8\_t procNum)  
*Locks the SEMA4 gate.*
- static void [SEMA4\\_Unlock](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Unlocks the SEMA4 gate.*
- static int32\_t [SEMA4\\_GetLockProc](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Gets the status of the SEMA4 gate.*
- status\_t [SEMA4\\_ResetGate](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Resets the SEMA4 gate to an unlocked status.*
- static status\_t [SEMA4\\_ResetAllGates](#) (SEMA4\_Type \*base)  
*Resets all SEMA4 gates to an unlocked status.*
- static void [SEMA4\\_EnableGateNotifyInterrupt](#) (SEMA4\_Type \*base, uint8\_t procNum, uint32\_t mask)  
*Enable the gate notification interrupt.*
- static void [SEMA4\\_DisableGateNotifyInterrupt](#) (SEMA4\_Type \*base, uint8\_t procNum, uint32\_t mask)  
*Disable the gate notification interrupt.*
- static uint32\_t [SEMA4\\_GetGateNotifyStatus](#) (SEMA4\_Type \*base, uint8\_t procNum)  
*Get the gate notification flags.*
- status\_t [SEMA4\\_ResetGateNotify](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Resets the SEMA4 gate IRQ notification.*
- static status\_t [SEMA4\\_ResetAllGateNotify](#) (SEMA4\_Type \*base)  
*Resets all SEMA4 gates IRQ notification.*

## Function Documentation

### Driver version

- #define [FSL\\_SEMA4\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*SEMA4 driver version.*

## 17.2 Macro Definition Documentation

### 17.2.1 #define SEMA4\_GATE\_NUM\_RESET\_ALL (64U)

## 17.3 Function Documentation

### 17.3.1 void SEMA4\_Init ( SEMA4\_Type \* *base* )

This function initializes the SEMA4 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA4\_ResetGate or SEMA4\_ResetAllGates function.

Parameters

<i>base</i>	SEMA4 peripheral base address.
-------------	--------------------------------

### 17.3.2 void SEMA4\_Deinit ( SEMA4\_Type \* *base* )

This function de-initializes the SEMA4 module. It only disables the clock.

Parameters

<i>base</i>	SEMA4 peripheral base address.
-------------	--------------------------------

### 17.3.3 status\_t SEMA4\_TryLock ( SEMA4\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *procNum* )

This function tries to lock the specific SEMA4 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

<i>base</i>	SEMA4 peripheral base address.
-------------	--------------------------------

<i>gateNum</i>	Gate number to lock.
<i>procNum</i>	Current processor number.

Return values

<i>kStatus_Success</i>	Lock the sema4 gate successfully.
<i>kStatus_Fail</i>	Sema4 gate has been locked by another processor.

### 17.3.4 void SEMA4\_Lock ( SEMA4\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *procNum* )

This function locks the specific SEMA4 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>gateNum</i>	Gate number to lock.
<i>procNum</i>	Current processor number.

### 17.3.5 static void SEMA4\_Unlock ( SEMA4\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function unlocks the specific SEMA4 gate. It only writes unlock value to the SEMA4 gate register. However, it does not check whether the SEMA4 gate is locked by the current processor or not. As a result, if the SEMA4 gate is not locked by the current processor, this function has no effect.

Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>gateNum</i>	Gate number to unlock.

### 17.3.6 static int32\_t SEMA4\_GetLockProc ( SEMA4\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function checks the lock status of a specific SEMA4 gate.

## Function Documentation

### Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>gateNum</i>	Gate number.

### Returns

Return -1 if the gate is unlocked, otherwise return the processor number which has locked the gate.

### 17.3.7 **status\_t SEMA4\_ResetGate ( SEMA4\_Type \* *base*, uint8\_t *gateNum* )**

This function resets a SEMA4 gate to an unlocked status.

### Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>gateNum</i>	Gate number.

### Return values

<i>kStatus_Success</i>	SEMA4 gate is reset successfully.
<i>kStatus_Fail</i>	Some other reset process is ongoing.

### 17.3.8 **static status\_t SEMA4\_ResetAllGates ( SEMA4\_Type \* *base* ) [inline], [static]**

This function resets all SEMA4 gate to an unlocked status.

### Parameters

<i>base</i>	SEMA4 peripheral base address.
-------------	--------------------------------

### Return values

<i>kStatus_Success</i>	SEMA4 is reset successfully.
------------------------	------------------------------

<i>kStatus_Fail</i>	Some other reset process is ongoing.
---------------------	--------------------------------------

### 17.3.9 static void SEMA4\_EnableGateNotifyInterrupt ( SEMA4\_Type \* *base*, uint8\_t *procNum*, uint32\_t *mask* ) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>procNum</i>	Current processor number.
<i>mask</i>	OR'ed value of the gate index, for example: (1<<0)   (1<<1) means gate 0 and gate 1.

### 17.3.10 static void SEMA4\_DisableGateNotifyInterrupt ( SEMA4\_Type \* *base*, uint8\_t *procNum*, uint32\_t *mask* ) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>procNum</i>	Current processor number.
<i>mask</i>	OR'ed value of the gate index, for example: (1<<0)   (1<<1) means gate 0 and gate 1.

### 17.3.11 static uint32\_t SEMA4\_GetGateNotifyStatus ( SEMA4\_Type \* *base*, uint8\_t *procNum* ) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle. The status flags are cleared automatically when the gate is locked by current core or locked again before the other core.

## Function Documentation

### Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>procNum</i>	Current processor number.

### Returns

OR'ed value of the gate index, for example:  $(1 \ll 0) \mid (1 \ll 1)$  means gate 0 and gate 1 flags are pending.

### 17.3.12 `status_t SEMA4_ResetGateNotify ( SEMA4_Type * base, uint8_t gateNum )`

This function resets a SEMA4 gate IRQ notification.

### Parameters

<i>base</i>	SEMA4 peripheral base address.
<i>gateNum</i>	Gate number.

### Return values

<i>kStatus_Success</i>	Reset successfully.
<i>kStatus_Fail</i>	Some other reset process is ongoing.

### 17.3.13 `static status_t SEMA4_ResetAllGateNotify ( SEMA4_Type * base ) [inline], [static]`

This function resets all SEMA4 gate IRQ notifications.

### Parameters

<i>base</i>	SEMA4 peripheral base address.
-------------	--------------------------------

### Return values



<i>kStatus_Success</i>	Reset successfully.
<i>kStatus_Fail</i>	Some other reset process is ongoing.



## Chapter 18

# TMU: Thermal Management Unit Driver

### 18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the thermal management unit (TMU) module of MCUXpresso SDK devices.

### 18.2 Typical use case

#### 18.2.1 Monitor and report Configuration

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

### Data Structures

- struct [tmu\\_threshold\\_config\\_t](#)  
*configuration for TMU threshold. [More...](#)*
- struct [tmu\\_interrupt\\_status\\_t](#)  
*TMU interrupt status. [More...](#)*
- struct [tmu\\_config\\_t](#)  
*Configuration for TMU module. [More...](#)*

### Macros

- #define [FSL\\_TMU\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 1))  
*TMU driver version.*

### Enumerations

- enum [\\_tmu\\_interrupt\\_enable](#) {  
    [kTMU\\_ImmediateTemperatureInterruptEnable](#),  
    [kTMU\\_AverageTemperatureInterruptEnable](#),  
    [kTMU\\_AverageTemperatureCriticalInterruptEnable](#) }  
*TMU interrupt enable.*
- enum [\\_tmu\\_interrupt\\_status\\_flags](#) {  
    [kTMU\\_ImmediateTemperatureStatusFlags](#) = [TMU\\_TIDR\\_ITTE\\_MASK](#),  
    [kTMU\\_AverageTemperatureStatusFlags](#) = [TMU\\_TIDR\\_ATTTE\\_MASK](#),  
    [kTMU\\_AverageTemperatureCriticalStatusFlags](#) }  
*TMU interrupt status flags.*
- enum [\\_tmu\\_status\\_flags](#) {  
    [kTMU\\_IntervalExceededStatusFlags](#) = [TMU\\_TSR\\_MIE\\_MASK](#),  
    [kTMU\\_OutOfLowRangeStatusFlags](#) = [TMU\\_TSR\\_ORL\\_MASK](#),  
    [kTMU\\_OutOfHighRangeStatusFlags](#) }

## Typical use case

- TMU status flags.*
- enum `tmu_average_low_pass_filter_t` {  
    `kTMU_AverageLowPassFilter1_0` = 0U,  
    `kTMU_AverageLowPassFilter0_5` = 1U,  
    `kTMU_AverageLowPassFilter0_25` = 2U,  
    `kTMU_AverageLowPassFilter0_125` = 3U }
- Average low pass filter setting.*

## Functions

- void `TMU_Init` (TMU\_Type \*base, const `tmu_config_t` \*config)  
*Enable the access to TMU registers and Initialize TMU module.*
- void `TMU_Deinit` (TMU\_Type \*base)  
*De-initialize TMU module and Disable the access to DCDC registers.*
- void `TMU_GetDefaultConfig` (`tmu_config_t` \*config)  
*Gets the default configuration for TMU.*
- static void `TMU_Enable` (TMU\_Type \*base, bool enable)  
*Enable/Disable the TMU module.*
- static void `TMU_EnableInterrupts` (TMU\_Type \*base, uint32\_t mask)  
*Enable the TMU interrupts.*
- static void `TMU_DisableInterrupts` (TMU\_Type \*base, uint32\_t mask)  
*Disable the TMU interrupts.*
- void `TMU_GetInterruptStatusFlags` (TMU\_Type \*base, `tmu_interrupt_status_t` \*status)  
*Get interrupt status flags.*
- void `TMU_ClearInterruptStatusFlags` (TMU\_Type \*base, uint32\_t mask)  
*Clear interrupt status flags and corresponding interrupt critical site capture register.*
- static uint32\_t `TMU_GetStatusFlags` (TMU\_Type \*base)  
*Get TMU status flags.*
- status\_t `TMU_GetHighestTemperature` (TMU\_Type \*base, uint32\_t \*temperature)  
*Get the highest temperature reached for any enabled monitored site within the temperature sensor range.*
- status\_t `TMU_GetLowestTemperature` (TMU\_Type \*base, uint32\_t \*temperature)  
*Get the lowest temperature reached for any enabled monitored site within the temperature sensor range.*
- status\_t `TMU_GetImmediateTemperature` (TMU\_Type \*base, uint32\_t siteIndex, uint32\_t \*temperature)  
*Get the last immediate temperature at site n.*
- status\_t `TMU_GetAverageTemperature` (TMU\_Type \*base, uint32\_t siteIndex, uint32\_t \*temperature)  
*Get the last average temperature at site n.*
- void `TMU_SetHighTemperatureThresold` (TMU\_Type \*base, const `tmu_thresold_config_t` \*config)  
*Configure the high temperature thresold value and enable/disable relevant thresold.*

## Variables

- bool `tmu_thresold_config_t::immediateThresoldEnable`  
*Enable high temperature immediate threshold.*
- bool `tmu_thresold_config_t::AverageThresoldEnable`  
*Enable high temperature average threshold.*
- bool `tmu_thresold_config_t::AverageCriticalThresoldEnable`  
*Enable high temperature average critical threshold.*
- uint8\_t `tmu_thresold_config_t::immediateThresoldValue`  
*Range:0U-125U.*

- uint8\_t [tmu\\_threshold\\_config\\_t::averageThresoldValue](#)  
*Range:0U-125U.*
- uint8\_t [tmu\\_threshold\\_config\\_t::averageCriticalThresoldValue](#)  
*Range:0U-125U.*
- uint32\_t [tmu\\_interrupt\\_status\\_t::interruptDetectMask](#)  
*The mask of interrupt status flags.*
- uint16\_t [tmu\\_interrupt\\_status\\_t::immediateInterruptsSiteMask](#)  
*The mask of the temperature sensor site associated with a detected ITTE event.*
- uint16\_t [tmu\\_interrupt\\_status\\_t::AverageInterruptsSiteMask](#)  
*The mask of the temperature sensor site associated with a detected ATTE event.*
- uint16\_t [tmu\\_interrupt\\_status\\_t::AverageCriticalInterruptsSiteMask](#)  
*The mask of the temperature sensor site associated with a detected ATCTE event.*
- uint8\_t [tmu\\_config\\_t::monitorInterval](#)  
*Temperature monitoring interval in seconds.*
- uint16\_t [tmu\\_config\\_t::monitorSiteSelection](#)  
*By setting the select bit for a temperature sensor site, it is enabled and included in all monitoring functions.*
- [tmu\\_average\\_low\\_pass\\_filter\\_t](#) [tmu\\_config\\_t::averageLPF](#)  
*The average temperature is calculated as:  $ALPF \times \text{Current\_Temp} + (1 - ALPF) \times \text{Average\_Temp}$ .*

## 18.3 Data Structure Documentation

### 18.3.1 struct tmu\_threshold\_config\_t

#### Data Fields

- bool [immediateThresoldEnable](#)  
*Enable high temperature immediate threshold.*
- bool [AverageThresoldEnable](#)  
*Enable high temperature average threshold.*
- bool [AverageCriticalThresoldEnable](#)  
*Enable high temperature average critical threshold.*
- uint8\_t [immediateThresoldValue](#)  
*Range:0U-125U.*
- uint8\_t [averageThresoldValue](#)  
*Range:0U-125U.*
- uint8\_t [averageCriticalThresoldValue](#)  
*Range:0U-125U.*

### 18.3.2 struct tmu\_interrupt\_status\_t

#### Data Fields

- uint32\_t [interruptDetectMask](#)  
*The mask of interrupt status flags.*
- uint16\_t [immediateInterruptsSiteMask](#)  
*The mask of the temperature sensor site associated with a detected ITTE event.*
- uint16\_t [AverageInterruptsSiteMask](#)  
*The mask of the temperature sensor site associated with a detected ATTE event.*

## Enumeration Type Documentation

- uint16\_t [AverageCriticalInterruptsSiteMask](#)  
*The mask of the temperature sensor site associated with a detected ATCTE event.*

### 18.3.3 struct tmu\_config\_t

#### Data Fields

- uint8\_t [monitorInterval](#)  
*Temperature monitoring interval in seconds.*
- uint16\_t [monitorSiteSelection](#)  
*By setting the select bit for a temperature sensor site, it is enabled and included in all monitoring functions.*
- [tmu\\_average\\_low\\_pass\\_filter\\_t](#) [averageLPF](#)  
*The average temperature is calculated as:  $ALPF \times Current\_Temp + (1 - ALPF) \times Average\_Temp$ .*

## 18.4 Macro Definition Documentation

### 18.4.1 #define FSL\_TMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

Version 2.0.1.

## 18.5 Enumeration Type Documentation

### 18.5.1 enum \_tmu\_interrupt\_enable

Enumerator

- kTMU\_ImmediateTemperatureInterruptEnable*** Immediate temperature threshold exceeded interrupt enable.
- kTMU\_AverageTemperatureInterruptEnable*** Average temperature threshold exceeded interrupt enable.
- kTMU\_AverageTemperatureCriticalInterruptEnable*** Average temperature critical threshold exceeded interrupt enable. >

### 18.5.2 enum \_tmu\_interrupt\_status\_flags

Enumerator

- kTMU\_ImmediateTemperatureStatusFlags*** Immediate temperature threshold exceeded(ITTE).
- kTMU\_AverageTemperatureStatusFlags*** Average temperature threshold exceeded(ATTE).
- kTMU\_AverageTemperatureCriticalStatusFlags*** Average temperature critical threshold exceeded. (ATCTE)

### 18.5.3 enum \_tmu\_status\_flags

Enumerator

***kTMU\_IntervalExceededStatusFlags*** Monitoring interval exceeded. The time required to perform measurement of all monitored sites has exceeded the monitoring interval as defined by TMTM-IR.

***kTMU\_OutOfLowRangeStatusFlags*** Out-of-range low temperature measurement detected. A temperature sensor detected a temperature reading below the lowest measurable temperature of 0 °C.

***kTMU\_OutOfHighRangeStatusFlags*** Out-of-range high temperature measurement detected. A temperature sensor detected a temperature reading above the highest measurable temperature of 125 °C.

### 18.5.4 enum tmu\_average\_low\_pass\_filter\_t

Enumerator

***kTMU\_AverageLowPassFilter1\_0*** Average low pass filter = 1.

***kTMU\_AverageLowPassFilter0\_5*** Average low pass filter = 0.5.

***kTMU\_AverageLowPassFilter0\_25*** Average low pass filter = 0.25.

***kTMU\_AverageLowPassFilter0\_125*** Average low pass filter = 0.125.

## 18.6 Function Documentation

### 18.6.1 void TMU\_Init ( TMU\_Type \* *base*, const tmu\_config\_t \* *config* )

Parameters

<i>base</i>	TMU peripheral base address.
<i>config</i>	Pointer to configuration structure. Refer to "tmu_config_t" structure.

### 18.6.2 void TMU\_Deinit ( TMU\_Type \* *base* )

Parameters

## Function Documentation

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

### 18.6.3 void TMU\_GetDefaultConfig ( tmu\_config\_t \* *config* )

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

Example:

```
config->monitorInterval = 0U;  
config->monitorSiteSelection = 0U;  
config->averageLPF = kTMU_AverageLowPassFilter1_0;
```

Parameters

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

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

Parameters

<i>base</i>	TMU peripheral base address.
<i>enable</i>	Switcher to enable/disable TMU.

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

Parameters

<i>bas</i>	TMU peripheral base address.
<i>mask</i>	The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration.

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



## Parameters

<i>bas</i>	TMU peripheral base address.
<i>mask</i>	The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration.

### 18.6.7 void TMU\_GetInterruptStatusFlags ( TMU\_Type \* *base*, tmu\_interrupt\_status\_t \* *status* )

## Parameters

<i>base</i>	TMU peripheral base address.
<i>status</i>	The pointer to interrupt status structure. Record the current interrupt status. Please refer to "tmu_interrupt_status_t" structure.

### 18.6.8 void TMU\_ClearInterruptStatusFlags ( TMU\_Type \* *base*, uint32\_t *mask* )

## Parameters

<i>base</i>	TMU peripheral base address.
<i>The</i>	mask of interrupt status flags. Refer to "_tmu_interrupt_status_flags" enumeration.

### 18.6.9 static uint32\_t TMU\_GetStatusFlags ( TMU\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

The mask of status flags. Refer to "\_tmu\_status\_flags" enumeration.

### 18.6.10 status\_t TMU\_GetHighestTemperature ( TMU\_Type \* *base*, uint32\_t \* *temperature* )

## Function Documentation

### Parameters

<i>base</i>	TMU peripheral base address.
<i>temperature</i>	Highest temperature recorded in degrees Celsius by any enabled monitored site.

### Returns

Execution status.

### Return values

<i>kStatus_Success</i>	Temperature reading is valid.
<i>kStatus_Fail</i>	Temperature reading is not valid due to no measured temperature within the sensor range of 0-125 °C for an enabled monitored site.

### 18.6.11 **status\_t** TMU\_GetLowestTemperature ( **TMU\_Type** \* *base*, **uint32\_t** \* *temperature* )

### Parameters

<i>base</i>	TMU peripheral base address.
<i>temperature</i>	Lowest temperature recorded in degrees Celsius by any enabled monitored site.

### Returns

Execution status.

### Return values

<i>kStatus_Success</i>	Temperature reading is valid.
<i>kStatus_Fail</i>	Temperature reading is not valid due to no measured temperature within the sensor range of 0-125 °C for an enabled monitored site.

### 18.6.12 **status\_t** TMU\_GetImmediateTemperature ( **TMU\_Type** \* *base*, **uint32\_t** *siteIndex*, **uint32\_t** \* *temperature* )

The site must be part of the list of enabled monitored sites as defined by monitorSiteSelection in "tmu\_config\_t" structure.

## Parameters

<i>base</i>	TMU peripheral base address.
<i>siteIndex</i>	The index of the site user want to read. 0U: site0 ~ 15U: site15.
<i>temperature</i>	Last immediate temperature reading at site n .

## Returns

Execution status.

## Return values

<i>kStatus_Success</i>	Temperature reading is valid.
<i>kStatus_Fail</i>	Temperature reading is not valid because temperature out of sensor range or first measurement still pending.

### 18.6.13 **status\_t** TMU\_GetAverageTemperature ( **TMU\_Type** \* *base*, **uint32\_t** *siteIndex*, **uint32\_t** \* *temperature* )

The site must be part of the list of enabled monitored sites as defined by monitorSiteSelection in "tmu\_config\_t" structure.

## Parameters

<i>base</i>	TMU peripheral base address.
<i>siteIndex</i>	The index of the site user want to read. 0U: site0 ~ 15U: site15.
<i>temperature</i>	Last average temperature reading at site n .

## Returns

Execution status.

## Return values

<i>kStatus_Success</i>	Temperature reading is valid.
------------------------	-------------------------------

## Variable Documentation

<i>kStatus_Fail</i>	Temperature reading is not valid because temperature out of sensor range or first measurement still pending.
---------------------	--

### 18.6.14 void TMU\_SetHighTemperatureThresold ( TMU\_Type \* *base*, const tmu\_threshold\_config\_t \* *config* )

Parameters

<i>base</i>	TMU peripheral base address.
<i>config</i>	Pointer to configuration structure. Refer to "tmu_threshold_config_t" structure.

## 18.7 Variable Documentation

### 18.7.1 bool tmu\_threshold\_config\_t::immediateThresoldEnable

### 18.7.2 bool tmu\_threshold\_config\_t::AverageThresoldEnable

### 18.7.3 bool tmu\_threshold\_config\_t::AverageCriticalThresoldEnable

### 18.7.4 uint8\_t tmu\_threshold\_config\_t::immediateThresoldValue

Valid when corresponding thresold is enabled. High temperature immediate threshold value. Determines the current upper temperature threshold, for anyenabled monitored site.

### 18.7.5 uint8\_t tmu\_threshold\_config\_t::averageThresoldValue

Valid when corresponding thresold is enabled. High temperature average threshold value. Determines the average upper temperature threshold, for any enabled monitored site.

### 18.7.6 uint8\_t tmu\_threshold\_config\_t::averageCriticalThresoldValue

Valid when corresponding thresold is enabled. High temperature average critical threshold value. Determines the average upper critical temperature threshold, for any enabled monitored site.

### 18.7.7 uint32\_t tmu\_interrupt\_status\_t::interruptDetectMask

Refer to "\_tmu\_interrupt\_status\_flags" enumeration.

**18.7.8 uint16\_t tmu\_interrupt\_status\_t::immediateInterruptsSiteMask**

Please refer to "\_tmu\_monitor\_site" enumeration.

**18.7.9 uint16\_t tmu\_interrupt\_status\_t::AverageInterruptsSiteMask**

Please refer to "\_tmu\_monitor\_site" enumeration.

**18.7.10 uint16\_t tmu\_interrupt\_status\_t::AverageCriticalInterruptsSiteMask**

Please refer to "\_tmu\_monitor\_site" enumeration.

**18.7.11 uint8\_t tmu\_config\_t::monitorInterval**

Please refer to specific table in RM.

**18.7.12 uint16\_t tmu\_config\_t::monitorSiteSelection**

If no site is selected, site 0 is monitored by default. Refer to "\_tmu\_monitor\_site" enumeration. Please look up relevant table in reference manual.

**18.7.13 tmu\_average\_low\_pass\_filter\_t tmu\_config\_t::averageLPF**

For proper operation, this field should only change when monitoring is disabled.



## Chapter 19

# WDOG: Watchdog Timer Driver

### 19.1 Overview

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

### 19.2 Typical use case

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

### Data Structures

- struct [wdog\\_work\\_mode\\_t](#)  
*Defines WDOG work mode. [More...](#)*
- struct [wdog\\_config\\_t](#)  
*Describes WDOG configuration structure. [More...](#)*

### Enumerations

- enum [\\_wdog\\_interrupt\\_enable](#) { [kWDOG\\_InterruptEnable](#) = WDOG\_WICR\_WIE\_MASK }
  - enum [\\_wdog\\_status\\_flags](#) {  
[kWDOG\\_RunningFlag](#) = WDOG\_WCR\_WDE\_MASK,  
[kWDOG\\_PowerOnResetFlag](#) = WDOG\_WRSR\_POR\_MASK,  
[kWDOG\\_TimeoutResetFlag](#) = WDOG\_WRSR\_TOUT\_MASK,  
[kWDOG\\_SoftwareResetFlag](#) = WDOG\_WRSR\_SFTW\_MASK,  
[kWDOG\\_InterruptFlag](#) = WDOG\_WICR\_WTIS\_MASK }
- WDOG status flags.*

### Driver version

- #define [FSL\\_WDOG\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 1, 0))  
*Defines WDOG driver version.*

### Refresh sequence

- #define [WDOG\\_REFRESH\\_KEY](#) (0xAAAA5555U)

### WDOG Initialization and De-initialization.

- void [WDOG\\_GetDefaultConfig](#) ([wdog\\_config\\_t](#) \*config)  
*Initializes the WDOG configuration structure.*
- void [WDOG\\_Init](#) (WDOG\_Type \*base, const [wdog\\_config\\_t](#) \*config)

## Data Structure Documentation

- Initializes the WDOG.*
- void [WDOG\\_Deinit](#) (WDOG\_Type \*base)
- Shuts down the WDOG.*
- static void [WDOG\\_Enable](#) (WDOG\_Type \*base)
- Enables the WDOG module.*
- static void [WDOG\\_Disable](#) (WDOG\_Type \*base)
- Disables the WDOG module.*
- static void [WDOG\\_TriggerSystemSoftwareReset](#) (WDOG\_Type \*base)
- Trigger the system software reset.*
- static void [WDOG\\_TriggerSoftwareSignal](#) (WDOG\_Type \*base)
- Trigger an output assertion.*
- static void [WDOG\\_EnableInterrupts](#) (WDOG\_Type \*base, uint16\_t mask)
- Enables the WDOG interrupt.*
- uint16\_t [WDOG\\_GetStatusFlags](#) (WDOG\_Type \*base)
- Gets the WDOG all reset status flags.*
- void [WDOG\\_ClearInterruptStatus](#) (WDOG\_Type \*base, uint16\_t mask)
- Clears the WDOG flag.*
- static void [WDOG\\_SetTimeoutValue](#) (WDOG\_Type \*base, uint16\_t timeoutCount)
- Sets the WDOG timeout value.*
- static void [WDOG\\_SetInterruptTimeoutValue](#) (WDOG\_Type \*base, uint16\_t timeoutCount)
- Sets the WDOG interrupt count timeout value.*
- static void [WDOG\\_DisablePowerDownEnable](#) (WDOG\_Type \*base)
- Disable the WDOG power down enable bit.*
- void [WDOG\\_Refresh](#) (WDOG\_Type \*base)
- Refreshes the WDOG timer.*

## 19.3 Data Structure Documentation

### 19.3.1 struct wdog\_work\_mode\_t

#### Data Fields

- bool [enableWait](#)  
*continue or suspend WDOG in wait mode*
- bool [enableStop](#)  
*continue or suspend WDOG in stop mode*
- bool [enableDebug](#)  
*continue or suspend WDOG in debug mode*

### 19.3.2 struct wdog\_config\_t

#### Data Fields

- bool [enableWdog](#)  
*Enables or disables WDOG.*
- [wdog\\_work\\_mode\\_t](#) [workMode](#)  
*Configures WDOG work mode in debug stop and wait mode.*
- bool [enableInterrupt](#)



- *Enables or disables WDOG interrupt.*  
uint16\_t **timeoutValue**  
*Timeout value.*
- uint16\_t **interruptTimeValue**  
*Interrupt count timeout value.*
- bool **softwareResetExtension**  
*software reset extension*
- bool **enablePowerDown**  
*power down enable bit*
- bool **enableTimeOutAssert**  
*Enable WDOG\_B timeout assertion.*

### 19.3.2.0.0.19 Field Documentation

#### 19.3.2.0.0.19.1 bool wdog\_config\_t::enableTimeOutAssert

## 19.4 Enumeration Type Documentation

### 19.4.1 enum \_wdog\_interrupt\_enable

This structure contains the settings for all of the WDOG interrupt configurations.

Enumerator

***kWDOG\_InterruptEnable*** WDOG timeout generates an interrupt before reset.

### 19.4.2 enum \_wdog\_status\_flags

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

***kWDOG\_RunningFlag*** Running flag, set when WDOG is enabled.

***kWDOG\_PowerOnResetFlag*** Power On flag, set when reset is the result of a powerOnReset.

***kWDOG\_TimeoutResetFlag*** Timeout flag, set when reset is the result of a timeout.

***kWDOG\_SoftwareResetFlag*** Software flag, set when reset is the result of a software.

***kWDOG\_InterruptFlag*** interrupt flag, whether interrupt has occurred or not

## 19.5 Function Documentation

### 19.5.1 void WDOG\_GetDefaultConfig ( wdog\_config\_t \* *config* )

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
* wdogConfig->enableWdog = true;
* wdogConfig->workMode.enableWait = true;
* wdogConfig->workMode.enableStop = false;
```

## Function Documentation

```
* wdogConfig->workMode.enableDebug = false;
* wdogConfig->enableInterrupt = false;
* wdogConfig->enablePowerdown = false;
* wdogConfig->resetExtension = false;
* wdogConfig->timeoutValue = 0xFFU;
* wdogConfig->interruptTimeValue = 0x04u;
*
```

### Parameters

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

### See Also

[wdog\\_config\\_t](#)

### 19.5.2 void WDOG\_Init ( WDOG\_Type \* *base*, const wdog\_config\_t \* *config* )

This function initializes the WDOG. When called, the WDOG runs according to the configuration.

This is an example.

```
* wdog_config_t config;
* WDOG_GetDefaultConfig(&config);
* config.timeoutValue = 0xffU;
* config->interruptTimeValue = 0x04u;
* WDOG_Init(wdog_base, &config);
*
```

### Parameters

<i>base</i>	WDOG peripheral base address
<i>config</i>	The configuration of WDOG

### 19.5.3 void WDOG\_Deinit ( WDOG\_Type \* *base* )

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

### 19.5.4 static void WDOG\_Enable ( WDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

Parameters

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

### 19.5.5 static void WDOG\_Disable ( WDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception

Parameters

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

### 19.5.6 static void WDOG\_TriggerSystemSoftwareReset ( WDOG\_Type \* *base* ) [inline], [static]

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to "1" after it has been asserted to "0". Note: Calling this API will reset the system right now, please using it with more attention.

Parameters

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

### 19.5.7 static void WDOG\_TriggerSoftwareSignal ( WDOG\_Type \* *base* ) [inline], [static]

This function will write to the WCR[WDA] bit to trigger WDOG\_B signal assertion. The WDOG\_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG\_B signal. Note: The WDOG\_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

Parameters

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

### 19.5.8 static void WDOG\_EnableInterrupts ( WDOG\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

## Parameters

<i>base</i>	WDOG peripheral base address
<i>mask</i>	The interrupts to enable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"> <li>• kWDOG_InterruptEnable</li> </ul>

### 19.5.9 uint16\_t WDOG\_GetStatusFlags ( WDOG\_Type \* *base* )

This function gets all reset status flags.

```
* uint16_t status;
* status = WDOG_GetStatusFlags (wdog_base);
*
```

## Parameters

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

## Returns

State of the status flag: asserted (true) or not-asserted (false).

## See Also

[\\_wdog\\_status\\_flags](#)

- true: a related status flag has been set.
- false: a related status flag is not set.

### 19.5.10 void WDOG\_ClearInterruptStatus ( WDOG\_Type \* *base*, uint16\_t *mask* )

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
* WDOG_ClearStatusFlags (wdog_base, kWDOG_InterruptFlag);
*
```

## Function Documentation

### 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

### 19.5.11 static void WDOG\_SetTimeoutValue ( WDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

### Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

### 19.5.12 static void WDOG\_SetInterruptTimeoutValue ( WDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are write-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

### Parameters

<i>base</i>	WDOG peripheral base address
<i>timeoutCount</i>	WDOG timeout value; count of WDOG clock tick.

### 19.5.13 static void WDOG\_DisablePowerDownEnable ( WDOG\_Type \* *base* ) [inline], [static]

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are write-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

## Parameters

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

**19.5.14 void WDOG\_Refresh ( WDOG\_Type \* *base* )**

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

## Parameters

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





## Chapter 20

# IOMUXC: Input/Output Multiplexing Controller

### 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Input/Output Multiplexing Controller.

The IOMUXC, working in conjunction with the IOMUX, enables the chip to share one pad for multiple signals from different peripheral interfaces.

### Files

- file [fsl\\_iomuxc.h](#)

### Driver version

- #define [FSL\\_IOMUXC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*IOMUXC driver version 2.0.0.*

### Pin function ID

- #define [IOMUXC\\_PMIC\\_STBY\\_REQ](#) 0x30330014, 0x0, 0x00000000, 0x0, 0x3033027C  
*The pin function ID is a tuple of <muxRegister muxmode="" inputregister="" inputdaisy="" configregister>="">*
- #define [IOMUXC\\_PMIC\\_ON\\_REQ](#) 0x30330018, 0x0, 0x00000000, 0x0, 0x30330280
- #define [IOMUXC\\_ONOFF](#) 0x3033001C, 0x0, 0x00000000, 0x0, 0x30330284
- #define [IOMUXC\\_POR\\_B](#) 0x30330020, 0x0, 0x00000000, 0x0, 0x30330288
- #define [IOMUXC\\_RTC\\_RESET\\_B](#) 0x30330024, 0x0, 0x00000000, 0x0, 0x3033028C
- #define [IOMUXC\\_GPIO1\\_IO00\\_GPIO1\\_IO00](#) 0x30330028, 0x0, 0x00000000, 0x0, 0x30330290
- #define [IOMUXC\\_GPIO1\\_IO00\\_CCM\\_ENET\\_PHY\\_REF\\_CLK\\_ROOT](#) 0x30330028, 0x1, 0x00000000, 0x0, 0x30330290
- #define [IOMUXC\\_GPIO1\\_IO00\\_XTALOSC\\_REF\\_CLK\\_32K](#) 0x30330028, 0x5, 0x00000000, 0x0, 0x30330290
- #define [IOMUXC\\_GPIO1\\_IO00\\_CCM\\_EXT\\_CLK1](#) 0x30330028, 0x6, 0x00000000, 0x0, 0x30330290
- #define [IOMUXC\\_GPIO1\\_IO01\\_GPIO1\\_IO01](#) 0x3033002C, 0x0, 0x00000000, 0x0, 0x30330294
- #define [IOMUXC\\_GPIO1\\_IO01\\_PWM1\\_OUT](#) 0x3033002C, 0x1, 0x00000000, 0x0, 0x30330294
- #define [IOMUXC\\_GPIO1\\_IO01\\_XTALOSC\\_REF\\_CLK\\_24M](#) 0x3033002C, 0x5, 0x00000000, 0x0, 0x30330294
- #define [IOMUXC\\_GPIO1\\_IO01\\_CCM\\_EXT\\_CLK2](#) 0x3033002C, 0x6, 0x00000000, 0x0, 0x30330294
- #define [IOMUXC\\_GPIO1\\_IO02\\_GPIO1\\_IO02](#) 0x30330030, 0x0, 0x00000000, 0x0, 0x30330298
- #define [IOMUXC\\_GPIO1\\_IO02\\_WDOG1\\_WDOG\\_B](#) 0x30330030, 0x1, 0x00000000, 0x0, 0x30330298
- #define [IOMUXC\\_GPIO1\\_IO02\\_WDOG1\\_WDOG\\_ANY](#) 0x30330030, 0x5, 0x00000000, 0x0, 0x30330298
- #define [IOMUXC\\_GPIO1\\_IO03\\_GPIO1\\_IO03](#) 0x30330034, 0x0, 0x00000000, 0x0, 0x3033029C

## Overview

- #define **IOMUXC\_GPIO1\_IO03\_USDHC1\_VSELECT** 0x30330034, 0x1, 0x00000000, 0x0, 0x3033029C
- #define **IOMUXC\_GPIO1\_IO03\_SDMA1\_EXT\_EVENT0** 0x30330034, 0x5, 0x00000000, 0x0, 0x3033029C
- #define **IOMUXC\_GPIO1\_IO04\_GPIO1\_IO04** 0x30330038, 0x0, 0x00000000, 0x0, 0x303302-A0
- #define **IOMUXC\_GPIO1\_IO04\_USDHC2\_VSELECT** 0x30330038, 0x1, 0x00000000, 0x0, 0x303302A0
- #define **IOMUXC\_GPIO1\_IO04\_SDMA1\_EXT\_EVENT1** 0x30330038, 0x5, 0x00000000, 0x0, 0x303302A0
- #define **IOMUXC\_GPIO1\_IO05\_GPIO1\_IO05** 0x3033003C, 0x0, 0x00000000, 0x0, 0x303302-A4
- #define **IOMUXC\_GPIO1\_IO05\_M4\_NMI** 0x3033003C, 0x1, 0x00000000, 0x0, 0x303302A4
- #define **IOMUXC\_GPIO1\_IO05\_CCM\_PMIC\_READY** 0x3033003C, 0x5, 0x303304BC, 0x0, 0x303302A4
- #define **IOMUXC\_GPIO1\_IO06\_GPIO1\_IO06** 0x30330040, 0x0, 0x00000000, 0x0, 0x303302-A8
- #define **IOMUXC\_GPIO1\_IO06\_ENET1\_MDC** 0x30330040, 0x1, 0x00000000, 0x0, 0x303302-A8
- #define **IOMUXC\_GPIO1\_IO06\_USDHC1\_CD\_B** 0x30330040, 0x5, 0x00000000, 0x0, 0x303302A8
- #define **IOMUXC\_GPIO1\_IO06\_CCM\_EXT\_CLK3** 0x30330040, 0x6, 0x00000000, 0x0, 0x303302A8
- #define **IOMUXC\_GPIO1\_IO07\_GPIO1\_IO07** 0x30330044, 0x0, 0x00000000, 0x0, 0x303302-AC
- #define **IOMUXC\_GPIO1\_IO07\_ENET1\_MDIO** 0x30330044, 0x1, 0x303304C0, 0x0, 0x303302AC
- #define **IOMUXC\_GPIO1\_IO07\_USDHC1\_WP** 0x30330044, 0x5, 0x00000000, 0x0, 0x303302-AC
- #define **IOMUXC\_GPIO1\_IO07\_CCM\_EXT\_CLK4** 0x30330044, 0x6, 0x00000000, 0x0, 0x303302AC
- #define **IOMUXC\_GPIO1\_IO08\_GPIO1\_IO08** 0x30330048, 0x0, 0x00000000, 0x0, 0x303302-B0
- #define **IOMUXC\_GPIO1\_IO08\_ENET1\_1588\_EVENT0\_IN** 0x30330048, 0x1, 0x00000000, 0x0, 0x303302B0
- #define **IOMUXC\_GPIO1\_IO08\_USDHC2\_RESET\_B** 0x30330048, 0x5, 0x00000000, 0x0, 0x303302B0
- #define **IOMUXC\_GPIO1\_IO09\_GPIO1\_IO09** 0x3033004C, 0x0, 0x00000000, 0x0, 0x303302-B4
- #define **IOMUXC\_GPIO1\_IO09\_ENET1\_1588\_EVENT0\_OUT** 0x3033004C, 0x1, 0x00000000, 0x0, 0x303302B4
- #define **IOMUXC\_GPIO1\_IO09\_SDMA2\_EXT\_EVENT0** 0x3033004C, 0x5, 0x00000000, 0x0, 0x303302B4
- #define **IOMUXC\_GPIO1\_IO10\_GPIO1\_IO10** 0x30330050, 0x0, 0x00000000, 0x0, 0x303302-B8
- #define **IOMUXC\_GPIO1\_IO10\_USB1\_OTG\_ID** 0x30330050, 0x1, 0x00000000, 0x0, 0x303302B8
- #define **IOMUXC\_GPIO1\_IO11\_GPIO1\_IO11** 0x30330054, 0x0, 0x00000000, 0x0, 0x303302-BC
- #define **IOMUXC\_GPIO1\_IO11\_USB2\_OTG\_ID** 0x30330054, 0x1, 0x00000000, 0x0,

- 0x303302BC
- #define **IOMUXC\_GPIO1\_IO11\_CCM\_PMIC\_READY** 0x30330054, 0x5, 0x303304BC, 0x1, 0x303302BC
- #define **IOMUXC\_GPIO1\_IO12\_GPIO1\_IO12** 0x30330058, 0x0, 0x00000000, 0x0, 0x303302-C0
- #define **IOMUXC\_GPIO1\_IO12\_USB1\_OTG\_PWR** 0x30330058, 0x1, 0x00000000, 0x0, 0x303302C0
- #define **IOMUXC\_GPIO1\_IO12\_SDMA2\_EXT\_EVENT1** 0x30330058, 0x5, 0x00000000, 0x0, 0x303302C0
- #define **IOMUXC\_GPIO1\_IO13\_GPIO1\_IO13** 0x3033005C, 0x0, 0x00000000, 0x0, 0x303302-C4
- #define **IOMUXC\_GPIO1\_IO13\_USB1\_OTG\_OC** 0x3033005C, 0x1, 0x00000000, 0x0, 0x303302C4
- #define **IOMUXC\_GPIO1\_IO13\_PWM2\_OUT** 0x3033005C, 0x5, 0x00000000, 0x0, 0x303302-C4
- #define **IOMUXC\_GPIO1\_IO14\_GPIO1\_IO14** 0x30330060, 0x0, 0x00000000, 0x0, 0x303302-C8
- #define **IOMUXC\_GPIO1\_IO14\_USB2\_OTG\_PWR** 0x30330060, 0x1, 0x00000000, 0x0, 0x303302C8
- #define **IOMUXC\_GPIO1\_IO14\_PWM3\_OUT** 0x30330060, 0x5, 0x00000000, 0x0, 0x303302-C8
- #define **IOMUXC\_GPIO1\_IO14\_CCM\_CLKO1** 0x30330060, 0x6, 0x00000000, 0x0, 0x303302-C8
- #define **IOMUXC\_GPIO1\_IO15\_GPIO1\_IO15** 0x30330064, 0x0, 0x00000000, 0x0, 0x303302-CC
- #define **IOMUXC\_GPIO1\_IO15\_USB2\_OTG\_OC** 0x30330064, 0x1, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC\_GPIO1\_IO15\_PWM4\_OUT** 0x30330064, 0x5, 0x00000000, 0x0, 0x303302-CC
- #define **IOMUXC\_GPIO1\_IO15\_CCM\_CLKO2** 0x30330064, 0x6, 0x00000000, 0x0, 0x303302-CC
- #define **IOMUXC\_ENET\_MDC\_ENET1\_MDC** 0x30330068, 0x0, 0x00000000, 0x0, 0x303302-D0
- #define **IOMUXC\_ENET\_MDC\_GPIO1\_IO16** 0x30330068, 0x5, 0x00000000, 0x0, 0x303302-D0
- #define **IOMUXC\_ENET\_MDIO\_ENET1\_MDIO** 0x3033006C, 0x0, 0x303304C0, 0x1, 0x303302D4
- #define **IOMUXC\_ENET\_MDIO\_GPIO1\_IO17** 0x3033006C, 0x5, 0x00000000, 0x0, 0x303302-D4
- #define **IOMUXC\_ENET\_TD3\_ENET1\_RGMII\_TD3** 0x30330070, 0x0, 0x00000000, 0x0, 0x303302D8
- #define **IOMUXC\_ENET\_TD3\_GPIO1\_IO18** 0x30330070, 0x5, 0x00000000, 0x0, 0x303302D8
- #define **IOMUXC\_ENET\_TD2\_ENET1\_RGMII\_TD2** 0x30330074, 0x0, 0x00000000, 0x0, 0x303302DC
- #define **IOMUXC\_ENET\_TD2\_ENET1\_TX\_CLK** 0x30330074, 0x1, 0x00000000, 0x0, 0x303302DC
- #define **IOMUXC\_ENET\_TD2\_GPIO1\_IO19** 0x30330074, 0x5, 0x00000000, 0x0, 0x303302DC
- #define **IOMUXC\_ENET\_TD1\_ENET1\_RGMII\_TD1** 0x30330078, 0x0, 0x00000000, 0x0, 0x303302E0
- #define **IOMUXC\_ENET\_TD1\_GPIO1\_IO20** 0x30330078, 0x5, 0x00000000, 0x0, 0x303302E0

## Overview

- **#define IOMUXC\_ENET\_TD0\_ENET1\_RGMII\_TD0** 0x3033007C, 0x0, 0x00000000, 0x0, 0x303302E4
- **#define IOMUXC\_ENET\_TD0\_GPIO1\_IO21** 0x3033007C, 0x5, 0x00000000, 0x0, 0x303302E4
- **#define IOMUXC\_ENET\_TX\_CTL\_ENET1\_RGMII\_TX\_CTL** 0x30330080, 0x0, 0x00000000, 0x0, 0x303302E8
- **#define IOMUXC\_ENET\_TX\_CTL\_GPIO1\_IO22** 0x30330080, 0x5, 0x00000000, 0x0, 0x303302E8
- **#define IOMUXC\_ENET\_TXC\_ENET1\_RGMII\_TXC** 0x30330084, 0x0, 0x00000000, 0x0, 0x303302EC
- **#define IOMUXC\_ENET\_TXC\_ENET1\_TX\_ER** 0x30330084, 0x1, 0x00000000, 0x0, 0x303302EC
- **#define IOMUXC\_ENET\_TXC\_GPIO1\_IO23** 0x30330084, 0x5, 0x00000000, 0x0, 0x303302EC
- **#define IOMUXC\_ENET\_RX\_CTL\_ENET1\_RGMII\_RX\_CTL** 0x30330088, 0x0, 0x00000000, 0x0, 0x303302F0
- **#define IOMUXC\_ENET\_RX\_CTL\_GPIO1\_IO24** 0x30330088, 0x5, 0x00000000, 0x0, 0x303302F0
- **#define IOMUXC\_ENET\_RXC\_ENET1\_RGMII\_RXC** 0x3033008C, 0x0, 0x00000000, 0x0, 0x303302F4
- **#define IOMUXC\_ENET\_RXC\_ENET1\_RX\_ER** 0x3033008C, 0x1, 0x00000000, 0x0, 0x303302F4
- **#define IOMUXC\_ENET\_RXC\_GPIO1\_IO25** 0x3033008C, 0x5, 0x00000000, 0x0, 0x303302F4
- **#define IOMUXC\_ENET\_RD0\_ENET1\_RGMII\_RD0** 0x30330090, 0x0, 0x00000000, 0x0, 0x303302F8
- **#define IOMUXC\_ENET\_RD0\_GPIO1\_IO26** 0x30330090, 0x5, 0x00000000, 0x0, 0x303302F8
- **#define IOMUXC\_ENET\_RD1\_ENET1\_RGMII\_RD1** 0x30330094, 0x0, 0x00000000, 0x0, 0x303302FC
- **#define IOMUXC\_ENET\_RD1\_GPIO1\_IO27** 0x30330094, 0x5, 0x00000000, 0x0, 0x303302FC
- **#define IOMUXC\_ENET\_RD2\_ENET1\_RGMII\_RD2** 0x30330098, 0x0, 0x00000000, 0x0, 0x30330300
- **#define IOMUXC\_ENET\_RD2\_GPIO1\_IO28** 0x30330098, 0x5, 0x00000000, 0x0, 0x30330300
- **#define IOMUXC\_ENET\_RD3\_ENET1\_RGMII\_RD3** 0x3033009C, 0x0, 0x00000000, 0x0, 0x30330304
- **#define IOMUXC\_ENET\_RD3\_GPIO1\_IO29** 0x3033009C, 0x5, 0x00000000, 0x0, 0x30330304
- **#define IOMUXC\_SD1\_CLK\_USDHC1\_CLK** 0x303300A0, 0x0, 0x00000000, 0x0, 0x30330308
- **#define IOMUXC\_SD1\_CLK\_GPIO2\_IO00** 0x303300A0, 0x5, 0x00000000, 0x0, 0x30330308
- **#define IOMUXC\_SD1\_CMD\_USDHC1\_CMD** 0x303300A4, 0x0, 0x00000000, 0x0, 0x3033030C
- **#define IOMUXC\_SD1\_CMD\_GPIO2\_IO01** 0x303300A4, 0x5, 0x00000000, 0x0, 0x3033030C
- **#define IOMUXC\_SD1\_DATA0\_USDHC1\_DATA0** 0x303300A8, 0x0, 0x00000000, 0x0, 0x30330310
- **#define IOMUXC\_SD1\_DATA0\_GPIO2\_IO02** 0x303300A8, 0x5, 0x00000000, 0x0, 0x30330310
- **#define IOMUXC\_SD1\_DATA1\_USDHC1\_DATA1** 0x303300AC, 0x0, 0x00000000, 0x0, 0x30330314
- **#define IOMUXC\_SD1\_DATA1\_GPIO2\_IO03** 0x303300AC, 0x5, 0x00000000, 0x0, 0x30330314
- **#define IOMUXC\_SD1\_DATA2\_USDHC1\_DATA2** 0x303300B0, 0x0, 0x00000000, 0x0, 0x30330318
- **#define IOMUXC\_SD1\_DATA2\_GPIO2\_IO04** 0x303300B0, 0x5, 0x00000000, 0x0, 0x30330318
- **#define IOMUXC\_SD1\_DATA3\_USDHC1\_DATA3** 0x303300B4, 0x0, 0x00000000, 0x0, 0x3033031C

- #define **IOMUXC\_SD1\_DATA3\_GPIO2\_IO05** 0x303300B4, 0x5, 0x00000000, 0x0, 0x3033031-C
- #define **IOMUXC\_SD1\_DATA4\_USDHC1\_DATA4** 0x303300B8, 0x0, 0x00000000, 0x0, 0x30330320
- #define **IOMUXC\_SD1\_DATA4\_GPIO2\_IO06** 0x303300B8, 0x5, 0x00000000, 0x0, 0x30330320
- #define **IOMUXC\_SD1\_DATA5\_USDHC1\_DATA5** 0x303300BC, 0x0, 0x00000000, 0x0, 0x30330324
- #define **IOMUXC\_SD1\_DATA5\_GPIO2\_IO07** 0x303300BC, 0x5, 0x00000000, 0x0, 0x30330324
- #define **IOMUXC\_SD1\_DATA6\_USDHC1\_DATA6** 0x303300C0, 0x0, 0x00000000, 0x0, 0x30330328
- #define **IOMUXC\_SD1\_DATA6\_GPIO2\_IO08** 0x303300C0, 0x5, 0x00000000, 0x0, 0x30330328
- #define **IOMUXC\_SD1\_DATA7\_USDHC1\_DATA7** 0x303300C4, 0x0, 0x00000000, 0x0, 0x3033032C
- #define **IOMUXC\_SD1\_DATA7\_GPIO2\_IO09** 0x303300C4, 0x5, 0x00000000, 0x0, 0x3033032-C
- #define **IOMUXC\_SD1\_RESET\_B\_USDHC1\_RESET\_B** 0x303300C8, 0x0, 0x00000000, 0x0, 0x30330330
- #define **IOMUXC\_SD1\_RESET\_B\_GPIO2\_IO10** 0x303300C8, 0x5, 0x00000000, 0x0, 0x30330330
- #define **IOMUXC\_SD1\_STROBE\_USDHC1\_STROBE** 0x303300CC, 0x0, 0x00000000, 0x0, 0x30330334
- #define **IOMUXC\_SD1\_STROBE\_GPIO2\_IO11** 0x303300CC, 0x5, 0x00000000, 0x0, 0x30330334
- #define **IOMUXC\_SD2\_CD\_B\_USDHC2\_CD\_B** 0x303300D0, 0x0, 0x00000000, 0x0, 0x30330338
- #define **IOMUXC\_SD2\_CD\_B\_GPIO2\_IO12** 0x303300D0, 0x5, 0x00000000, 0x0, 0x30330338
- #define **IOMUXC\_SD2\_CLK\_USDHC2\_CLK** 0x303300D4, 0x0, 0x00000000, 0x0, 0x3033033-C
- #define **IOMUXC\_SD2\_CLK\_GPIO2\_IO13** 0x303300D4, 0x5, 0x00000000, 0x0, 0x3033033C
- #define **IOMUXC\_SD2\_CMD\_USDHC2\_CMD** 0x303300D8, 0x0, 0x00000000, 0x0, 0x30330340
- #define **IOMUXC\_SD2\_CMD\_GPIO2\_IO14** 0x303300D8, 0x5, 0x00000000, 0x0, 0x30330340
- #define **IOMUXC\_SD2\_DATA0\_USDHC2\_DATA0** 0x303300DC, 0x0, 0x00000000, 0x0, 0x30330344
- #define **IOMUXC\_SD2\_DATA0\_GPIO2\_IO15** 0x303300DC, 0x5, 0x00000000, 0x0, 0x30330344
- #define **IOMUXC\_SD2\_DATA1\_USDHC2\_DATA1** 0x303300E0, 0x0, 0x00000000, 0x0, 0x30330348
- #define **IOMUXC\_SD2\_DATA1\_GPIO2\_IO16** 0x303300E0, 0x5, 0x00000000, 0x0, 0x30330348
- #define **IOMUXC\_SD2\_DATA2\_USDHC2\_DATA2** 0x303300E4, 0x0, 0x00000000, 0x0, 0x3033034C
- #define **IOMUXC\_SD2\_DATA2\_GPIO2\_IO17** 0x303300E4, 0x5, 0x00000000, 0x0, 0x3033034-C
- #define **IOMUXC\_SD2\_DATA3\_USDHC2\_DATA3** 0x303300E8, 0x0, 0x00000000, 0x0, 0x30330350
- #define **IOMUXC\_SD2\_DATA3\_GPIO2\_IO18** 0x303300E8, 0x5, 0x00000000, 0x0, 0x30330350
- #define **IOMUXC\_SD2\_RESET\_B\_USDHC2\_RESET\_B** 0x303300EC, 0x0, 0x00000000, 0x0, 0x30330354
- #define **IOMUXC\_SD2\_RESET\_B\_GPIO2\_IO19** 0x303300EC, 0x5, 0x00000000, 0x0, 0x30330354
- #define **IOMUXC\_SD2\_WP\_USDHC2\_WP** 0x303300F0, 0x0, 0x00000000, 0x0, 0x30330358
- #define **IOMUXC\_SD2\_WP\_GPIO2\_IO20** 0x303300F0, 0x5, 0x00000000, 0x0, 0x30330358
- #define **IOMUXC\_NAND\_ALE\_RAWNAND\_ALE** 0x303300F4, 0x0, 0x00000000, 0x0,

## Overview

- 0x3033035C
- #define **IOMUXC\_NAND\_ALE\_QSPI\_A\_SCLK** 0x303300F4, 0x1, 0x00000000, 0x0, 0x3033035C
- #define **IOMUXC\_NAND\_ALE\_GPIO3\_IO00** 0x303300F4, 0x5, 0x00000000, 0x0, 0x3033035C
- #define **IOMUXC\_NAND\_CE0\_B\_RAWNAND\_CE0\_B** 0x303300F8, 0x0, 0x00000000, 0x0, 0x30330360
- #define **IOMUXC\_NAND\_CE0\_B\_QSPI\_A\_SS0\_B** 0x303300F8, 0x1, 0x00000000, 0x0, 0x30330360
- #define **IOMUXC\_NAND\_CE0\_B\_GPIO3\_IO01** 0x303300F8, 0x5, 0x00000000, 0x0, 0x30330360
- #define **IOMUXC\_NAND\_CE1\_B\_RAWNAND\_CE1\_B** 0x303300FC, 0x0, 0x00000000, 0x0, 0x30330364
- #define **IOMUXC\_NAND\_CE1\_B\_QSPI\_A\_SS1\_B** 0x303300FC, 0x1, 0x00000000, 0x0, 0x30330364
- #define **IOMUXC\_NAND\_CE1\_B\_GPIO3\_IO02** 0x303300FC, 0x5, 0x00000000, 0x0, 0x30330364
- #define **IOMUXC\_NAND\_CE2\_B\_RAWNAND\_CE2\_B** 0x30330100, 0x0, 0x00000000, 0x0, 0x30330368
- #define **IOMUXC\_NAND\_CE2\_B\_QSPI\_B\_SS0\_B** 0x30330100, 0x1, 0x00000000, 0x0, 0x30330368
- #define **IOMUXC\_NAND\_CE2\_B\_GPIO3\_IO03** 0x30330100, 0x5, 0x00000000, 0x0, 0x30330368
- #define **IOMUXC\_NAND\_CE3\_B\_RAWNAND\_CE3\_B** 0x30330104, 0x0, 0x00000000, 0x0, 0x3033036C
- #define **IOMUXC\_NAND\_CE3\_B\_QSPI\_B\_SS1\_B** 0x30330104, 0x1, 0x00000000, 0x0, 0x3033036C
- #define **IOMUXC\_NAND\_CE3\_B\_GPIO3\_IO04** 0x30330104, 0x5, 0x00000000, 0x0, 0x3033036C
- #define **IOMUXC\_NAND\_CLE\_RAWNAND\_CLE** 0x30330108, 0x0, 0x00000000, 0x0, 0x30330370
- #define **IOMUXC\_NAND\_CLE\_QSPI\_B\_SCLK** 0x30330108, 0x1, 0x00000000, 0x0, 0x30330370
- #define **IOMUXC\_NAND\_CLE\_GPIO3\_IO05** 0x30330108, 0x5, 0x00000000, 0x0, 0x30330370
- #define **IOMUXC\_NAND\_DATA00\_RAWNAND\_DATA00** 0x3033010C, 0x0, 0x00000000, 0x0, 0x30330374
- #define **IOMUXC\_NAND\_DATA00\_QSPI\_A\_DATA0** 0x3033010C, 0x1, 0x00000000, 0x0, 0x30330374
- #define **IOMUXC\_NAND\_DATA00\_GPIO3\_IO06** 0x3033010C, 0x5, 0x00000000, 0x0, 0x30330374
- #define **IOMUXC\_NAND\_DATA01\_RAWNAND\_DATA01** 0x30330110, 0x0, 0x00000000, 0x0, 0x30330378
- #define **IOMUXC\_NAND\_DATA01\_QSPI\_A\_DATA1** 0x30330110, 0x1, 0x00000000, 0x0, 0x30330378
- #define **IOMUXC\_NAND\_DATA01\_GPIO3\_IO07** 0x30330110, 0x5, 0x00000000, 0x0, 0x30330378
- #define **IOMUXC\_NAND\_DATA02\_RAWNAND\_DATA02** 0x30330114, 0x0, 0x00000000, 0x0, 0x3033037C
- #define **IOMUXC\_NAND\_DATA02\_QSPI\_A\_DATA2** 0x30330114, 0x1, 0x00000000, 0x0, 0x3033037C

- **#define IOMUXC\_NAND\_DATA02\_GPIO3\_IO08** 0x30330114, 0x5, 0x00000000, 0x0, 0x3033037C
- **#define IOMUXC\_NAND\_DATA03\_RAWNAND\_DATA03** 0x30330118, 0x0, 0x00000000, 0x0, 0x30330380
- **#define IOMUXC\_NAND\_DATA03\_QSPI\_A\_DATA3** 0x30330118, 0x1, 0x00000000, 0x0, 0x30330380
- **#define IOMUXC\_NAND\_DATA03\_GPIO3\_IO09** 0x30330118, 0x5, 0x00000000, 0x0, 0x30330380
- **#define IOMUXC\_NAND\_DATA04\_RAWNAND\_DATA04** 0x3033011C, 0x0, 0x00000000, 0x0, 0x30330384
- **#define IOMUXC\_NAND\_DATA04\_QSPI\_B\_DATA0** 0x3033011C, 0x1, 0x00000000, 0x0, 0x30330384
- **#define IOMUXC\_NAND\_DATA04\_GPIO3\_IO10** 0x3033011C, 0x5, 0x00000000, 0x0, 0x30330384
- **#define IOMUXC\_NAND\_DATA05\_RAWNAND\_DATA05** 0x30330120, 0x0, 0x00000000, 0x0, 0x30330388
- **#define IOMUXC\_NAND\_DATA05\_QSPI\_B\_DATA1** 0x30330120, 0x1, 0x00000000, 0x0, 0x30330388
- **#define IOMUXC\_NAND\_DATA05\_GPIO3\_IO11** 0x30330120, 0x5, 0x00000000, 0x0, 0x30330388
- **#define IOMUXC\_NAND\_DATA06\_RAWNAND\_DATA06** 0x30330124, 0x0, 0x00000000, 0x0, 0x3033038C
- **#define IOMUXC\_NAND\_DATA06\_QSPI\_B\_DATA2** 0x30330124, 0x1, 0x00000000, 0x0, 0x3033038C
- **#define IOMUXC\_NAND\_DATA06\_GPIO3\_IO12** 0x30330124, 0x5, 0x00000000, 0x0, 0x3033038C
- **#define IOMUXC\_NAND\_DATA07\_RAWNAND\_DATA07** 0x30330128, 0x0, 0x00000000, 0x0, 0x30330390
- **#define IOMUXC\_NAND\_DATA07\_QSPI\_B\_DATA3** 0x30330128, 0x1, 0x00000000, 0x0, 0x30330390
- **#define IOMUXC\_NAND\_DATA07\_GPIO3\_IO13** 0x30330128, 0x5, 0x00000000, 0x0, 0x30330390
- **#define IOMUXC\_NAND\_DQS\_RAWNAND\_DQS** 0x3033012C, 0x0, 0x00000000, 0x0, 0x30330394
- **#define IOMUXC\_NAND\_DQS\_QSPI\_A\_DQS** 0x3033012C, 0x1, 0x00000000, 0x0, 0x30330394
- **#define IOMUXC\_NAND\_DQS\_GPIO3\_IO14** 0x3033012C, 0x5, 0x00000000, 0x0, 0x30330394
- **#define IOMUXC\_NAND\_RE\_B\_RAWNAND\_RE\_B** 0x30330130, 0x0, 0x00000000, 0x0, 0x30330398
- **#define IOMUXC\_NAND\_RE\_B\_QSPI\_B\_DQS** 0x30330130, 0x1, 0x00000000, 0x0, 0x30330398
- **#define IOMUXC\_NAND\_RE\_B\_GPIO3\_IO15** 0x30330130, 0x5, 0x00000000, 0x0, 0x30330398
- **#define IOMUXC\_NAND\_READY\_B\_RAWNAND\_READY\_B** 0x30330134, 0x0, 0x00000000, 0x0, 0x3033039C
- **#define IOMUXC\_NAND\_READY\_B\_GPIO3\_IO16** 0x30330134, 0x5, 0x00000000, 0x0, 0x3033039C
- **#define IOMUXC\_NAND\_WE\_B\_RAWNAND\_WE\_B** 0x30330138, 0x0, 0x00000000, 0x0, 0x303303A0
- **#define IOMUXC\_NAND\_WE\_B\_GPIO3\_IO17** 0x30330138, 0x5, 0x00000000, 0x0, 0x303303A0
- **#define IOMUXC\_NAND\_WP\_B\_RAWNAND\_WP\_B** 0x3033013C, 0x0, 0x00000000, 0x0, 0x303303A4

## Overview

- `#define IOMUXC_NAND_WP_B_GPIO3_IO18` 0x3033013C, 0x5, 0x00000000, 0x0, 0x303303-A4
- `#define IOMUXC_SAI5_RXFS_SAI5_RX_SYNC` 0x30330140, 0x0, 0x303304E4, 0x0, 0x303303A8
- `#define IOMUXC_SAI5_RXFS_SAI1_TX_DATA0` 0x30330140, 0x1, 0x00000000, 0x0, 0x303303A8
- `#define IOMUXC_SAI5_RXFS_GPIO3_IO19` 0x30330140, 0x5, 0x00000000, 0x0, 0x303303A8
- `#define IOMUXC_SAI5_RXC_SAI5_RX_BCLK` 0x30330144, 0x0, 0x303304D0, 0x0, 0x303303AC
- `#define IOMUXC_SAI5_RXC_SAI1_TX_DATA1` 0x30330144, 0x1, 0x00000000, 0x0, 0x303303AC
- `#define IOMUXC_SAI5_RXC_GPIO3_IO20` 0x30330144, 0x5, 0x00000000, 0x0, 0x303303AC
- `#define IOMUXC_SAI5_RXD0_SAI5_RX_DATA0` 0x30330148, 0x0, 0x303304D4, 0x0, 0x303303B0
- `#define IOMUXC_SAI5_RXD0_SAI1_TX_DATA2` 0x30330148, 0x1, 0x00000000, 0x0, 0x303303B0
- `#define IOMUXC_SAI5_RXD0_GPIO3_IO21` 0x30330148, 0x5, 0x00000000, 0x0, 0x303303B0
- `#define IOMUXC_SAI5_RXD1_SAI5_RX_DATA1` 0x3033014C, 0x0, 0x303304D8, 0x0, 0x303303B4
- `#define IOMUXC_SAI5_RXD1_SAI1_TX_DATA3` 0x3033014C, 0x1, 0x00000000, 0x0, 0x303303B4
- `#define IOMUXC_SAI5_RXD1_SAI1_TX_SYNC` 0x3033014C, 0x2, 0x303304CC, 0x0, 0x303303B4
- `#define IOMUXC_SAI5_RXD1_SAI5_TX_SYNC` 0x3033014C, 0x3, 0x303304EC, 0x0, 0x303303B4
- `#define IOMUXC_SAI5_RXD1_GPIO3_IO22` 0x3033014C, 0x5, 0x00000000, 0x0, 0x303303-B4
- `#define IOMUXC_SAI5_RXD2_SAI5_RX_DATA2` 0x30330150, 0x0, 0x303304DC, 0x0, 0x303303B8
- `#define IOMUXC_SAI5_RXD2_SAI1_TX_DATA4` 0x30330150, 0x1, 0x00000000, 0x0, 0x303303B8
- `#define IOMUXC_SAI5_RXD2_SAI1_TX_SYNC` 0x30330150, 0x2, 0x303304CC, 0x1, 0x303303B8
- `#define IOMUXC_SAI5_RXD2_SAI5_TX_BCLK` 0x30330150, 0x3, 0x303304E8, 0x0, 0x303303B8
- `#define IOMUXC_SAI5_RXD2_GPIO3_IO23` 0x30330150, 0x5, 0x00000000, 0x0, 0x303303B8
- `#define IOMUXC_SAI5_RXD3_SAI5_RX_DATA3` 0x30330154, 0x0, 0x303304E0, 0x0, 0x303303BC
- `#define IOMUXC_SAI5_RXD3_SAI1_TX_DATA5` 0x30330154, 0x1, 0x00000000, 0x0, 0x303303BC
- `#define IOMUXC_SAI5_RXD3_SAI1_TX_SYNC` 0x30330154, 0x2, 0x303304CC, 0x2, 0x303303BC
- `#define IOMUXC_SAI5_RXD3_SAI5_TX_DATA0` 0x30330154, 0x3, 0x00000000, 0x0, 0x303303BC
- `#define IOMUXC_SAI5_RXD3_GPIO3_IO24` 0x30330154, 0x5, 0x00000000, 0x0, 0x303303B-C
- `#define IOMUXC_SAI5_MCLK_SAI5_MCLK` 0x30330158, 0x0, 0x3033052C, 0x0, 0x303303-C0
- `#define IOMUXC_SAI5_MCLK_SAI1_TX_BCLK` 0x30330158, 0x1, 0x303304C8, 0x0, 0x303303C0



- #define **IOMUXC\_SAI5\_MCLK\_SAI4\_MCLK** 0x30330158, 0x2, 0x00000000, 0x0, 0x303303C0
- #define **IOMUXC\_SAI5\_MCLK\_GPIO3\_IO25** 0x30330158, 0x5, 0x00000000, 0x0, 0x303303C0
- #define **IOMUXC\_SAI1\_RXFS\_SAI1\_RX\_SYNC** 0x3033015C, 0x0, 0x303304C4, 0x0, 0x303303C4
- #define **IOMUXC\_SAI1\_RXFS\_SAI5\_RX\_SYNC** 0x3033015C, 0x1, 0x303304E4, 0x1, 0x303303C4
- #define **IOMUXC\_SAI1\_RXFS\_CORESIGHT\_TRACE\_CLK** 0x3033015C, 0x4, 0x00000000, 0x0, 0x303303C4
- #define **IOMUXC\_SAI1\_RXFS\_GPIO4\_IO00** 0x3033015C, 0x5, 0x00000000, 0x0, 0x303303C4
- #define **IOMUXC\_SAI1\_RXC\_SAI1\_RX\_BCLK** 0x30330160, 0x0, 0x00000000, 0x0, 0x303303C8
- #define **IOMUXC\_SAI1\_RXC\_SAI5\_RX\_BCLK** 0x30330160, 0x1, 0x303304D0, 0x1, 0x303303C8
- #define **IOMUXC\_SAI1\_RXC\_CORESIGHT\_TRACE\_CTL** 0x30330160, 0x4, 0x00000000, 0x0, 0x303303C8
- #define **IOMUXC\_SAI1\_RXC\_GPIO4\_IO01** 0x30330160, 0x5, 0x00000000, 0x0, 0x303303C8
- #define **IOMUXC\_SAI1\_RXD0\_SAI1\_RX\_DATA0** 0x30330164, 0x0, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC\_SAI1\_RXD0\_SAI5\_RX\_DATA0** 0x30330164, 0x1, 0x303304D4, 0x1, 0x303303CC
- #define **IOMUXC\_SAI1\_RXD0\_CORESIGHT\_TRACE0** 0x30330164, 0x4, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC\_SAI1\_RXD0\_GPIO4\_IO02** 0x30330164, 0x5, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC\_SAI1\_RXD0\_SRC\_BOOT\_CFG0** 0x30330164, 0x6, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC\_SAI1\_RXD1\_SAI1\_RX\_DATA1** 0x30330168, 0x0, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC\_SAI1\_RXD1\_SAI5\_RX\_DATA1** 0x30330168, 0x1, 0x303304D8, 0x1, 0x303303D0
- #define **IOMUXC\_SAI1\_RXD1\_CORESIGHT\_TRACE1** 0x30330168, 0x4, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC\_SAI1\_RXD1\_GPIO4\_IO03** 0x30330168, 0x5, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC\_SAI1\_RXD1\_SRC\_BOOT\_CFG1** 0x30330168, 0x6, 0x00000000, 0x0, 0x303303D0
- #define **IOMUXC\_SAI1\_RXD2\_SAI1\_RX\_DATA2** 0x3033016C, 0x0, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC\_SAI1\_RXD2\_SAI5\_RX\_DATA2** 0x3033016C, 0x1, 0x303304DC, 0x1, 0x303303D4
- #define **IOMUXC\_SAI1\_RXD2\_CORESIGHT\_TRACE2** 0x3033016C, 0x4, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC\_SAI1\_RXD2\_GPIO4\_IO04** 0x3033016C, 0x5, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC\_SAI1\_RXD2\_SRC\_BOOT\_CFG2** 0x3033016C, 0x6, 0x00000000, 0x0, 0x303303D4
- #define **IOMUXC\_SAI1\_RXD3\_SAI1\_RX\_DATA3** 0x30330170, 0x0, 0x00000000, 0x0, 0x303303D8

## Overview

- `#define IOMUXC_SAI1_RXD3_SAI5_RX_DATA3 0x30330170, 0x1, 0x303304E0, 0x1, 0x303303D8`
- `#define IOMUXC_SAI1_RXD3_CORESIGHT_TRACE3 0x30330170, 0x4, 0x00000000, 0x0, 0x303303D8`
- `#define IOMUXC_SAI1_RXD3_GPIO4_IO05 0x30330170, 0x5, 0x00000000, 0x0, 0x303303D8`
- `#define IOMUXC_SAI1_RXD3_SRC_BOOT_CFG3 0x30330170, 0x6, 0x00000000, 0x0, 0x303303D8`
- `#define IOMUXC_SAI1_RXD4_SAI1_RX_DATA4 0x30330174, 0x0, 0x00000000, 0x0, 0x303303DC`
- `#define IOMUXC_SAI1_RXD4_SAI6_TX_BCLK 0x30330174, 0x1, 0x3033051C, 0x0, 0x303303DC`
- `#define IOMUXC_SAI1_RXD4_SAI6_RX_BCLK 0x30330174, 0x2, 0x30330510, 0x0, 0x303303DC`
- `#define IOMUXC_SAI1_RXD4_CORESIGHT_TRACE4 0x30330174, 0x4, 0x00000000, 0x0, 0x303303DC`
- `#define IOMUXC_SAI1_RXD4_GPIO4_IO06 0x30330174, 0x5, 0x00000000, 0x0, 0x303303DC`
- `#define IOMUXC_SAI1_RXD4_SRC_BOOT_CFG4 0x30330174, 0x6, 0x00000000, 0x0, 0x303303DC`
- `#define IOMUXC_SAI1_RXD5_SAI1_RX_DATA5 0x30330178, 0x0, 0x00000000, 0x0, 0x303303E0`
- `#define IOMUXC_SAI1_RXD5_SAI6_TX_DATA0 0x30330178, 0x1, 0x00000000, 0x0, 0x303303E0`
- `#define IOMUXC_SAI1_RXD5_SAI6_RX_DATA0 0x30330178, 0x2, 0x30330514, 0x0, 0x303303E0`
- `#define IOMUXC_SAI1_RXD5_SAI1_RX_SYNC 0x30330178, 0x3, 0x303304C4, 0x1, 0x303303E0`
- `#define IOMUXC_SAI1_RXD5_CORESIGHT_TRACE5 0x30330178, 0x4, 0x00000000, 0x0, 0x303303E0`
- `#define IOMUXC_SAI1_RXD5_GPIO4_IO07 0x30330178, 0x5, 0x00000000, 0x0, 0x303303E0`
- `#define IOMUXC_SAI1_RXD5_SRC_BOOT_CFG5 0x30330178, 0x6, 0x00000000, 0x0, 0x303303E0`
- `#define IOMUXC_SAI1_RXD6_SAI1_RX_DATA6 0x3033017C, 0x0, 0x00000000, 0x0, 0x303303E4`
- `#define IOMUXC_SAI1_RXD6_SAI6_TX_SYNC 0x3033017C, 0x1, 0x30330520, 0x0, 0x303303E4`
- `#define IOMUXC_SAI1_RXD6_SAI6_RX_SYNC 0x3033017C, 0x2, 0x30330518, 0x0, 0x303303E4`
- `#define IOMUXC_SAI1_RXD6_CORESIGHT_TRACE6 0x3033017C, 0x4, 0x00000000, 0x0, 0x303303E4`
- `#define IOMUXC_SAI1_RXD6_GPIO4_IO08 0x3033017C, 0x5, 0x00000000, 0x0, 0x303303E4`
- `#define IOMUXC_SAI1_RXD6_SRC_BOOT_CFG6 0x3033017C, 0x6, 0x00000000, 0x0, 0x303303E4`
- `#define IOMUXC_SAI1_RXD7_SAI1_RX_DATA7 0x30330180, 0x0, 0x00000000, 0x0, 0x303303E8`
- `#define IOMUXC_SAI1_RXD7_SAI6_MCLK 0x30330180, 0x1, 0x30330530, 0x0, 0x303303E8`
- `#define IOMUXC_SAI1_RXD7_SAI1_TX_SYNC 0x30330180, 0x2, 0x303304CC, 0x4, 0x303303E8`
- `#define IOMUXC_SAI1_RXD7_SAI1_TX_DATA4 0x30330180, 0x3, 0x00000000, 0x0, 0x303303E8`

- 0x303303E8
- #define **IOMUXC\_SAI1\_RXD7\_CORESIGHT\_TRACE7** 0x30330180, 0x4, 0x00000000, 0x0, 0x303303E8
- #define **IOMUXC\_SAI1\_RXD7\_GPIO4\_IO09** 0x30330180, 0x5, 0x00000000, 0x0, 0x303303E8
- #define **IOMUXC\_SAI1\_RXD7\_SRC\_BOOT\_CFG7** 0x30330180, 0x6, 0x00000000, 0x0, 0x303303E8
- #define **IOMUXC\_SAI1\_TXFS\_SAI1\_TX\_SYNC** 0x30330184, 0x0, 0x303304CC, 0x3, 0x303303EC
- #define **IOMUXC\_SAI1\_TXFS\_SAI5\_TX\_SYNC** 0x30330184, 0x1, 0x303304EC, 0x1, 0x303303EC
- #define **IOMUXC\_SAI1\_TXFS\_CORESIGHT\_EVENT0** 0x30330184, 0x4, 0x00000000, 0x0, 0x303303EC
- #define **IOMUXC\_SAI1\_TXFS\_GPIO4\_IO10** 0x30330184, 0x5, 0x00000000, 0x0, 0x303303EC
- #define **IOMUXC\_SAI1\_TXC\_SAI1\_TX\_BCLK** 0x30330188, 0x0, 0x303304C8, 0x1, 0x303303F0
- #define **IOMUXC\_SAI1\_TXC\_SAI5\_TX\_BCLK** 0x30330188, 0x1, 0x303304E8, 0x1, 0x303303F0
- #define **IOMUXC\_SAI1\_TXC\_CORESIGHT\_EVENT1** 0x30330188, 0x4, 0x00000000, 0x0, 0x303303F0
- #define **IOMUXC\_SAI1\_TXC\_GPIO4\_IO11** 0x30330188, 0x5, 0x00000000, 0x0, 0x303303F0
- #define **IOMUXC\_SAI1\_TXD0\_SAI1\_TX\_DATA0** 0x3033018C, 0x0, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC\_SAI1\_TXD0\_SAI5\_TX\_DATA0** 0x3033018C, 0x1, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC\_SAI1\_TXD0\_CORESIGHT\_TRACE8** 0x3033018C, 0x4, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC\_SAI1\_TXD0\_GPIO4\_IO12** 0x3033018C, 0x5, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC\_SAI1\_TXD0\_SRC\_BOOT\_CFG8** 0x3033018C, 0x6, 0x00000000, 0x0, 0x303303F4
- #define **IOMUXC\_SAI1\_TXD1\_SAI1\_TX\_DATA1** 0x30330190, 0x0, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC\_SAI1\_TXD1\_SAI5\_TX\_DATA1** 0x30330190, 0x1, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC\_SAI1\_TXD1\_CORESIGHT\_TRACE9** 0x30330190, 0x4, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC\_SAI1\_TXD1\_GPIO4\_IO13** 0x30330190, 0x5, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC\_SAI1\_TXD1\_SRC\_BOOT\_CFG9** 0x30330190, 0x6, 0x00000000, 0x0, 0x303303F8
- #define **IOMUXC\_SAI1\_TXD2\_SAI1\_TX\_DATA2** 0x30330194, 0x0, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC\_SAI1\_TXD2\_SAI5\_TX\_DATA2** 0x30330194, 0x1, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC\_SAI1\_TXD2\_CORESIGHT\_TRACE10** 0x30330194, 0x4, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC\_SAI1\_TXD2\_GPIO4\_IO14** 0x30330194, 0x5, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC\_SAI1\_TXD2\_SRC\_BOOT\_CFG10** 0x30330194, 0x6, 0x00000000, 0x0, 0x303303FC
- #define **IOMUXC\_SAI1\_TXD3\_SAI1\_TX\_DATA3** 0x30330198, 0x0, 0x00000000, 0x0, 0x30330400
- #define **IOMUXC\_SAI1\_TXD3\_SAI5\_TX\_DATA3** 0x30330198, 0x1, 0x00000000, 0x0, 0x30330400

## Overview

```
0x30330400
• #define IOMUXC_SAI1_TXD3_CORESIGHT_TRACE11 0x30330198, 0x4, 0x00000000, 0x0,
0x30330400
• #define IOMUXC_SAI1_TXD3_GPIO4_IO15 0x30330198, 0x5, 0x00000000, 0x0, 0x30330400
• #define IOMUXC_SAI1_TXD3_SRC_BOOT_CFG11 0x30330198, 0x6, 0x00000000, 0x0,
0x30330400
• #define IOMUXC_SAI1_TXD4_SAI1_TX_DATA4 0x3033019C, 0x0, 0x00000000, 0x0,
0x30330404
• #define IOMUXC_SAI1_TXD4_SAI6_RX_BCLK 0x3033019C, 0x1, 0x30330510, 0x1,
0x30330404
• #define IOMUXC_SAI1_TXD4_SAI6_TX_BCLK 0x3033019C, 0x2, 0x3033051C, 0x1,
0x30330404
• #define IOMUXC_SAI1_TXD4_CORESIGHT_TRACE12 0x3033019C, 0x4, 0x00000000, 0x0,
0x30330404
• #define IOMUXC_SAI1_TXD4_GPIO4_IO16 0x3033019C, 0x5, 0x00000000, 0x0, 0x30330404
• #define IOMUXC_SAI1_TXD4_SRC_BOOT_CFG12 0x3033019C, 0x6, 0x00000000, 0x0,
0x30330404
• #define IOMUXC_SAI1_TXD5_SAI1_TX_DATA5 0x303301A0, 0x0, 0x00000000, 0x0,
0x30330408
• #define IOMUXC_SAI1_TXD5_SAI6_RX_DATA0 0x303301A0, 0x1, 0x30330514, 0x1,
0x30330408
• #define IOMUXC_SAI1_TXD5_SAI6_TX_DATA0 0x303301A0, 0x2, 0x00000000, 0x0,
0x30330408
• #define IOMUXC_SAI1_TXD5_CORESIGHT_TRACE13 0x303301A0, 0x4, 0x00000000,
0x0, 0x30330408
• #define IOMUXC_SAI1_TXD5_GPIO4_IO17 0x303301A0, 0x5, 0x00000000, 0x0, 0x30330408
• #define IOMUXC_SAI1_TXD5_SRC_BOOT_CFG13 0x303301A0, 0x6, 0x00000000, 0x0,
0x30330408
• #define IOMUXC_SAI1_TXD6_SAI1_TX_DATA6 0x303301A4, 0x0, 0x00000000, 0x0,
0x3033040C
• #define IOMUXC_SAI1_TXD6_SAI6_RX_SYNC 0x303301A4, 0x1, 0x30330518, 0x1,
0x3033040C
• #define IOMUXC_SAI1_TXD6_SAI6_TX_SYNC 0x303301A4, 0x2, 0x30330520, 0x1,
0x3033040C
• #define IOMUXC_SAI1_TXD6_CORESIGHT_TRACE14 0x303301A4, 0x4, 0x00000000,
0x0, 0x3033040C
• #define IOMUXC_SAI1_TXD6_GPIO4_IO18 0x303301A4, 0x5, 0x00000000, 0x0, 0x3033040-
C
• #define IOMUXC_SAI1_TXD6_SRC_BOOT_CFG14 0x303301A4, 0x6, 0x00000000, 0x0,
0x3033040C
• #define IOMUXC_SAI1_TXD7_SAI1_TX_DATA7 0x303301A8, 0x0, 0x00000000, 0x0,
0x30330410
• #define IOMUXC_SAI1_TXD7_SAI6_MCLK 0x303301A8, 0x1, 0x30330530, 0x1, 0x30330410
• #define IOMUXC_SAI1_TXD7_CORESIGHT_TRACE15 0x303301A8, 0x4, 0x00000000,
0x0, 0x30330410
• #define IOMUXC_SAI1_TXD7_GPIO4_IO19 0x303301A8, 0x5, 0x00000000, 0x0, 0x30330410
• #define IOMUXC_SAI1_TXD7_SRC_BOOT_CFG15 0x303301A8, 0x6, 0x00000000, 0x0,
0x30330410
• #define IOMUXC_SAI1_MCLK_SAI1_MCLK 0x303301AC, 0x0, 0x00000000, 0x0, 0x30330414
• #define IOMUXC_SAI1_MCLK_SAI5_MCLK 0x303301AC, 0x1, 0x3033052C, 0x1, 0x30330414
• #define IOMUXC_SAI1_MCLK_SAI1_TX_BCLK 0x303301AC, 0x2, 0x303304C8, 0x2,
```

```

0x30330414
• #define IOMUXC_SAI1_MCLK_GPIO4_IO20 0x303301AC, 0x5, 0x00000000, 0x0, 0x30330414
• #define IOMUXC_SAI2_RXFS_SAI2_RX_SYNC 0x303301B0, 0x0, 0x00000000, 0x0,
0x30330418
• #define IOMUXC_SAI2_RXFS_SAI5_TX_SYNC 0x303301B0, 0x1, 0x303304EC, 0x2,
0x30330418
• #define IOMUXC_SAI2_RXFS_GPIO4_IO21 0x303301B0, 0x5, 0x00000000, 0x0, 0x30330418
• #define IOMUXC_SAI2_RXC_SAI2_RX_BCLK 0x303301B4, 0x0, 0x00000000, 0x0,
0x3033041C
• #define IOMUXC_SAI2_RXC_SAI5_TX_BCLK 0x303301B4, 0x1, 0x303304E8, 0x2,
0x3033041C
• #define IOMUXC_SAI2_RXC_GPIO4_IO22 0x303301B4, 0x5, 0x00000000, 0x0, 0x3033041C
• #define IOMUXC_SAI2_RXD0_SAI2_RX_DATA0 0x303301B8, 0x0, 0x00000000, 0x0,
0x30330420
• #define IOMUXC_SAI2_RXD0_SAI5_TX_DATA0 0x303301B8, 0x1, 0x00000000, 0x0,
0x30330420
• #define IOMUXC_SAI2_RXD0_GPIO4_IO23 0x303301B8, 0x5, 0x00000000, 0x0, 0x30330420
• #define IOMUXC_SAI2_TXFS_SAI2_TX_SYNC 0x303301BC, 0x0, 0x00000000, 0x0,
0x30330424
• #define IOMUXC_SAI2_TXFS_SAI5_TX_DATA1 0x303301BC, 0x1, 0x00000000, 0x0,
0x30330424
• #define IOMUXC_SAI2_TXFS_GPIO4_IO24 0x303301BC, 0x5, 0x00000000, 0x0, 0x30330424
• #define IOMUXC_SAI2_TXC_SAI2_TX_BCLK 0x303301C0, 0x0, 0x00000000, 0x0,
0x30330428
• #define IOMUXC_SAI2_TXC_SAI5_TX_DATA2 0x303301C0, 0x1, 0x00000000, 0x0,
0x30330428
• #define IOMUXC_SAI2_TXC_GPIO4_IO25 0x303301C0, 0x5, 0x00000000, 0x0, 0x30330428
• #define IOMUXC_SAI2_TXD0_SAI2_TX_DATA0 0x303301C4, 0x0, 0x00000000, 0x0,
0x3033042C
• #define IOMUXC_SAI2_TXD0_SAI5_TX_DATA3 0x303301C4, 0x1, 0x00000000, 0x0,
0x3033042C
• #define IOMUXC_SAI2_TXD0_GPIO4_IO26 0x303301C4, 0x5, 0x00000000, 0x0, 0x3033042-
C
• #define IOMUXC_SAI2_MCLK_SAI2_MCLK 0x303301C8, 0x0, 0x00000000, 0x0, 0x30330430
• #define IOMUXC_SAI2_MCLK_SAI5_MCLK 0x303301C8, 0x1, 0x3033052C, 0x2, 0x30330430
• #define IOMUXC_SAI2_MCLK_GPIO4_IO27 0x303301C8, 0x5, 0x00000000, 0x0, 0x30330430
• #define IOMUXC_SAI3_RXFS_SAI3_RX_SYNC 0x303301CC, 0x0, 0x00000000, 0x0,
0x30330434
• #define IOMUXC_SAI3_RXFS_GPT1_CAPTURE1 0x303301CC, 0x1, 0x00000000, 0x0,
0x30330434
• #define IOMUXC_SAI3_RXFS_SAI5_RX_SYNC 0x303301CC, 0x2, 0x303304E4, 0x2,
0x30330434
• #define IOMUXC_SAI3_RXFS_GPIO4_IO28 0x303301CC, 0x5, 0x00000000, 0x0, 0x30330434
• #define IOMUXC_SAI3_RXC_SAI3_RX_BCLK 0x303301D0, 0x0, 0x00000000, 0x0,
0x30330438
• #define IOMUXC_SAI3_RXC_GPT1_CAPTURE2 0x303301D0, 0x1, 0x00000000, 0x0,
0x30330438
• #define IOMUXC_SAI3_RXC_SAI5_RX_BCLK 0x303301D0, 0x2, 0x303304D0, 0x2,
0x30330438
• #define IOMUXC_SAI3_RXC_GPIO4_IO29 0x303301D0, 0x5, 0x00000000, 0x0, 0x30330438
• #define IOMUXC_SAI3_RXD_SAI3_RX_DATA0 0x303301D4, 0x0, 0x00000000, 0x0,

```

## Overview

- 0x3033043C
- #define **IOMUXC\_SAI3\_RXD\_GPT1\_COMPARE1** 0x303301D4, 0x1, 0x00000000, 0x0, 0x3033043C
- #define **IOMUXC\_SAI3\_RXD\_SAI5\_RX\_DATA0** 0x303301D4, 0x2, 0x303304D4, 0x2, 0x3033043C
- #define **IOMUXC\_SAI3\_RXD\_GPIO4\_IO30** 0x303301D4, 0x5, 0x00000000, 0x0, 0x3033043C
- #define **IOMUXC\_SAI3\_TXFS\_SAI3\_TX\_SYNC** 0x303301D8, 0x0, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_GPT1\_CLK** 0x303301D8, 0x1, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_SAI5\_RX\_DATA1** 0x303301D8, 0x2, 0x303304D8, 0x2, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_GPIO4\_IO31** 0x303301D8, 0x5, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC\_SAI3\_TXC\_SAI3\_TX\_BCLK** 0x303301DC, 0x0, 0x00000000, 0x0, 0x30330444
- #define **IOMUXC\_SAI3\_TXC\_GPT1\_COMPARE2** 0x303301DC, 0x1, 0x00000000, 0x0, 0x30330444
- #define **IOMUXC\_SAI3\_TXC\_SAI5\_RX\_DATA2** 0x303301DC, 0x2, 0x303304DC, 0x2, 0x30330444
- #define **IOMUXC\_SAI3\_TXC\_GPIO5\_IO00** 0x303301DC, 0x5, 0x00000000, 0x0, 0x30330444
- #define **IOMUXC\_SAI3\_TXD\_SAI3\_TX\_DATA0** 0x303301E0, 0x0, 0x00000000, 0x0, 0x30330448
- #define **IOMUXC\_SAI3\_TXD\_GPT1\_COMPARE3** 0x303301E0, 0x1, 0x00000000, 0x0, 0x30330448
- #define **IOMUXC\_SAI3\_TXD\_SAI5\_RX\_DATA3** 0x303301E0, 0x2, 0x303304E0, 0x2, 0x30330448
- #define **IOMUXC\_SAI3\_TXD\_GPIO5\_IO01** 0x303301E0, 0x5, 0x00000000, 0x0, 0x30330448
- #define **IOMUXC\_SAI3\_MCLK\_SAI3\_MCLK** 0x303301E4, 0x0, 0x00000000, 0x0, 0x3033044C
- #define **IOMUXC\_SAI3\_MCLK\_PWM4\_OUT** 0x303301E4, 0x1, 0x00000000, 0x0, 0x3033044C
- #define **IOMUXC\_SAI3\_MCLK\_SAI5\_MCLK** 0x303301E4, 0x2, 0x3033052C, 0x3, 0x3033044C
- #define **IOMUXC\_SAI3\_MCLK\_GPIO5\_IO02** 0x303301E4, 0x5, 0x00000000, 0x0, 0x3033044C
- #define **IOMUXC\_SPDIF\_TX\_SPDIF1\_OUT** 0x303301E8, 0x0, 0x00000000, 0x0, 0x30330450
- #define **IOMUXC\_SPDIF\_TX\_PWM3\_OUT** 0x303301E8, 0x1, 0x00000000, 0x0, 0x30330450
- #define **IOMUXC\_SPDIF\_TX\_GPIO5\_IO03** 0x303301E8, 0x5, 0x00000000, 0x0, 0x30330450
- #define **IOMUXC\_SPDIF\_RX\_SPDIF1\_IN** 0x303301EC, 0x0, 0x00000000, 0x0, 0x30330454
- #define **IOMUXC\_SPDIF\_RX\_PWM2\_OUT** 0x303301EC, 0x1, 0x00000000, 0x0, 0x30330454
- #define **IOMUXC\_SPDIF\_RX\_GPIO5\_IO04** 0x303301EC, 0x5, 0x00000000, 0x0, 0x30330454
- #define **IOMUXC\_SPDIF\_EXT\_CLK\_SPDIF1\_EXT\_CLK** 0x303301F0, 0x0, 0x00000000, 0x0, 0x30330458
- #define **IOMUXC\_SPDIF\_EXT\_CLK\_PWM1\_OUT** 0x303301F0, 0x1, 0x00000000, 0x0, 0x30330458
- #define **IOMUXC\_SPDIF\_EXT\_CLK\_GPIO5\_IO05** 0x303301F0, 0x5, 0x00000000, 0x0, 0x30330458
- #define **IOMUXC\_ECSP11\_SCLK\_ECSP11\_SCLK** 0x303301F4, 0x0, 0x00000000, 0x0, 0x3033045C
- #define **IOMUXC\_ECSP11\_SCLK\_UART3\_RX** 0x303301F4, 0x1, 0x30330504, 0x0, 0x3033045C
- #define **IOMUXC\_ECSP11\_SCLK\_UART3\_TX** 0x303301F4, 0x1, 0x00000000, 0x0,

- 0x3033045C
- #define **IOMUXC\_ECSP11\_SCLK\_GPIO5\_IO06** 0x303301F4, 0x5, 0x00000000, 0x0, 0x3033045C
- #define **IOMUXC\_ECSP11\_MOSI\_ECSP11\_MOSI** 0x303301F8, 0x0, 0x00000000, 0x0, 0x30330460
- #define **IOMUXC\_ECSP11\_MOSI\_UART3\_TX** 0x303301F8, 0x1, 0x00000000, 0x0, 0x30330460
- #define **IOMUXC\_ECSP11\_MOSI\_UART3\_RX** 0x303301F8, 0x1, 0x30330504, 0x1, 0x30330460
- #define **IOMUXC\_ECSP11\_MOSI\_GPIO5\_IO07** 0x303301F8, 0x5, 0x00000000, 0x0, 0x30330460
- #define **IOMUXC\_ECSP11\_MISO\_ECSP11\_MISO** 0x303301FC, 0x0, 0x00000000, 0x0, 0x30330464
- #define **IOMUXC\_ECSP11\_MISO\_UART3\_CTS\_B** 0x303301FC, 0x1, 0x00000000, 0x0, 0x30330464
- #define **IOMUXC\_ECSP11\_MISO\_UART3\_RTS\_B** 0x303301FC, 0x1, 0x30330500, 0x0, 0x30330464
- #define **IOMUXC\_ECSP11\_MISO\_GPIO5\_IO08** 0x303301FC, 0x5, 0x00000000, 0x0, 0x30330464
- #define **IOMUXC\_ECSP11\_SS0\_ECSP11\_SS0** 0x30330200, 0x0, 0x00000000, 0x0, 0x30330468
- #define **IOMUXC\_ECSP11\_SS0\_UART3\_RTS\_B** 0x30330200, 0x1, 0x30330500, 0x1, 0x30330468
- #define **IOMUXC\_ECSP11\_SS0\_UART3\_CTS\_B** 0x30330200, 0x1, 0x00000000, 0x0, 0x30330468
- #define **IOMUXC\_ECSP11\_SS0\_GPIO5\_IO09** 0x30330200, 0x5, 0x00000000, 0x0, 0x30330468
- #define **IOMUXC\_ECSP12\_SCLK\_ECSP12\_SCLK** 0x30330204, 0x0, 0x00000000, 0x0, 0x3033046C
- #define **IOMUXC\_ECSP12\_SCLK\_UART4\_RX** 0x30330204, 0x1, 0x3033050C, 0x0, 0x3033046C
- #define **IOMUXC\_ECSP12\_SCLK\_UART4\_TX** 0x30330204, 0x1, 0x00000000, 0x0, 0x3033046C
- #define **IOMUXC\_ECSP12\_SCLK\_GPIO5\_IO10** 0x30330204, 0x5, 0x00000000, 0x0, 0x3033046C
- #define **IOMUXC\_ECSP12\_MOSI\_ECSP12\_MOSI** 0x30330208, 0x0, 0x00000000, 0x0, 0x30330470
- #define **IOMUXC\_ECSP12\_MOSI\_UART4\_TX** 0x30330208, 0x1, 0x00000000, 0x0, 0x30330470
- #define **IOMUXC\_ECSP12\_MOSI\_UART4\_RX** 0x30330208, 0x1, 0x3033050C, 0x1, 0x30330470
- #define **IOMUXC\_ECSP12\_MOSI\_GPIO5\_IO11** 0x30330208, 0x5, 0x00000000, 0x0, 0x30330470
- #define **IOMUXC\_ECSP12\_MISO\_ECSP12\_MISO** 0x3033020C, 0x0, 0x00000000, 0x0, 0x30330474
- #define **IOMUXC\_ECSP12\_MISO\_UART4\_CTS\_B** 0x3033020C, 0x1, 0x00000000, 0x0, 0x30330474
- #define **IOMUXC\_ECSP12\_MISO\_UART4\_RTS\_B** 0x3033020C, 0x1, 0x30330508, 0x0, 0x30330474
- #define **IOMUXC\_ECSP12\_MISO\_GPIO5\_IO12** 0x3033020C, 0x5, 0x00000000, 0x0, 0x30330474
- #define **IOMUXC\_ECSP12\_SS0\_ECSP12\_SS0** 0x30330210, 0x0, 0x00000000, 0x0, 0x30330478
- #define **IOMUXC\_ECSP12\_SS0\_UART4\_RTS\_B** 0x30330210, 0x1, 0x30330508, 0x1, 0x30330478
- #define **IOMUXC\_ECSP12\_SS0\_UART4\_CTS\_B** 0x30330210, 0x1, 0x00000000, 0x0, 0x30330478

## Overview

```
0x30330478
• #define IOMUXC_ECSPi2_SS0_GPIO5_IO13 0x30330210, 0x5, 0x00000000, 0x0, 0x30330478
• #define IOMUXC_I2C1_SCL_I2C1_SCL 0x30330214, 0x0, 0x00000000, 0x0, 0x3033047C
• #define IOMUXC_I2C1_SCL_ENET1_MDC 0x30330214, 0x1, 0x00000000, 0x0, 0x3033047C
• #define IOMUXC_I2C1_SCL_GPIO5_IO14 0x30330214, 0x5, 0x00000000, 0x0, 0x3033047C
• #define IOMUXC_I2C1_SDA_I2C1_SDA 0x30330218, 0x0, 0x00000000, 0x0, 0x30330480
• #define IOMUXC_I2C1_SDA_ENET1_MDIO 0x30330218, 0x1, 0x303304C0, 0x2, 0x30330480
• #define IOMUXC_I2C1_SDA_GPIO5_IO15 0x30330218, 0x5, 0x00000000, 0x0, 0x30330480
• #define IOMUXC_I2C2_SCL_I2C2_SCL 0x3033021C, 0x0, 0x00000000, 0x0, 0x30330484
• #define IOMUXC_I2C2_SCL_ENET1_1588_EVENT1_IN 0x3033021C, 0x1, 0x00000000, 0x0,
0x30330484
• #define IOMUXC_I2C2_SCL_GPIO5_IO16 0x3033021C, 0x5, 0x00000000, 0x0, 0x30330484
• #define IOMUXC_I2C2_SDA_I2C2_SDA 0x30330220, 0x0, 0x00000000, 0x0, 0x30330488
• #define IOMUXC_I2C2_SDA_ENET1_1588_EVENT1_OUT 0x30330220, 0x1, 0x00000000,
0x0, 0x30330488
• #define IOMUXC_I2C2_SDA_GPIO5_IO17 0x30330220, 0x5, 0x00000000, 0x0, 0x30330488
• #define IOMUXC_I2C3_SCL_I2C3_SCL 0x30330224, 0x0, 0x00000000, 0x0, 0x3033048C
• #define IOMUXC_I2C3_SCL_PWM4_OUT 0x30330224, 0x1, 0x00000000, 0x0, 0x3033048C
• #define IOMUXC_I2C3_SCL_GPT2_CLK 0x30330224, 0x2, 0x00000000, 0x0, 0x3033048C
• #define IOMUXC_I2C3_SCL_GPIO5_IO18 0x30330224, 0x5, 0x00000000, 0x0, 0x3033048C
• #define IOMUXC_I2C3_SDA_I2C3_SDA 0x30330228, 0x0, 0x00000000, 0x0, 0x30330490
• #define IOMUXC_I2C3_SDA_PWM3_OUT 0x30330228, 0x1, 0x00000000, 0x0, 0x30330490
• #define IOMUXC_I2C3_SDA_GPT3_CLK 0x30330228, 0x2, 0x00000000, 0x0, 0x30330490
• #define IOMUXC_I2C3_SDA_GPIO5_IO19 0x30330228, 0x5, 0x00000000, 0x0, 0x30330490
• #define IOMUXC_I2C4_SCL_I2C4_SCL 0x3033022C, 0x0, 0x00000000, 0x0, 0x30330494
• #define IOMUXC_I2C4_SCL_PWM2_OUT 0x3033022C, 0x1, 0x00000000, 0x0, 0x30330494
• #define IOMUXC_I2C4_SCL_PCIE1_CLKREQ_B 0x3033022C, 0x2, 0x30330524, 0x0,
0x30330494
• #define IOMUXC_I2C4_SCL_GPIO5_IO20 0x3033022C, 0x5, 0x00000000, 0x0, 0x30330494
• #define IOMUXC_I2C4_SDA_I2C4_SDA 0x30330230, 0x0, 0x00000000, 0x0, 0x30330498
• #define IOMUXC_I2C4_SDA_PWM1_OUT 0x30330230, 0x1, 0x00000000, 0x0, 0x30330498
• #define IOMUXC_I2C4_SDA_PCIE2_CLKREQ_B 0x30330230, 0x2, 0x30330528, 0x0,
0x30330498
• #define IOMUXC_I2C4_SDA_GPIO5_IO21 0x30330230, 0x5, 0x00000000, 0x0, 0x30330498
• #define IOMUXC_UART1_RXD_UART1_RX 0x30330234, 0x0, 0x303304F4, 0x0, 0x3033049-
C
• #define IOMUXC_UART1_RXD_UART1_TX 0x30330234, 0x0, 0x00000000, 0x0, 0x3033049-
C
• #define IOMUXC_UART1_RXD_ECSPi3_SCLK 0x30330234, 0x1, 0x00000000, 0x0,
0x3033049C
• #define IOMUXC_UART1_RXD_GPIO5_IO22 0x30330234, 0x5, 0x00000000, 0x0, 0x3033049-
C
• #define IOMUXC_UART1_TXD_UART1_TX 0x30330238, 0x0, 0x00000000, 0x0, 0x303304-
A0
• #define IOMUXC_UART1_TXD_UART1_RX 0x30330238, 0x0, 0x303304F4, 0x1, 0x303304-
A0
• #define IOMUXC_UART1_TXD_ECSPi3_MOSI 0x30330238, 0x1, 0x00000000, 0x0,
0x303304A0
• #define IOMUXC_UART1_TXD_GPIO5_IO23 0x30330238, 0x5, 0x00000000, 0x0, 0x303304-
A0
• #define IOMUXC_UART2_RXD_UART2_RX 0x3033023C, 0x0, 0x303304FC, 0x0, 0x303304-
A4
• #define IOMUXC_UART2_RXD_UART2_TX 0x3033023C, 0x0, 0x00000000, 0x0, 0x303304-
```



- A4
- #define **IOMUXC\_UART2\_RXD\_ECSPi3\_MISO** 0x3033023C, 0x1, 0x00000000, 0x0, 0x303304A4
- #define **IOMUXC\_UART2\_RXD\_GPIO5\_IO24** 0x3033023C, 0x5, 0x00000000, 0x0, 0x303304A4
- #define **IOMUXC\_UART2\_TXD\_UART2\_TX** 0x30330240, 0x0, 0x00000000, 0x0, 0x303304A8
- #define **IOMUXC\_UART2\_TXD\_UART2\_RX** 0x30330240, 0x0, 0x303304FC, 0x1, 0x303304A8
- #define **IOMUXC\_UART2\_TXD\_ECSPi3\_SS0** 0x30330240, 0x1, 0x00000000, 0x0, 0x303304A8
- #define **IOMUXC\_UART2\_TXD\_GPIO5\_IO25** 0x30330240, 0x5, 0x00000000, 0x0, 0x303304A8
- #define **IOMUXC\_UART3\_RXD\_UART3\_RX** 0x30330244, 0x0, 0x30330504, 0x2, 0x303304AC
- #define **IOMUXC\_UART3\_RXD\_UART3\_TX** 0x30330244, 0x0, 0x00000000, 0x0, 0x303304AC
- #define **IOMUXC\_UART3\_RXD\_UART1\_CTS\_B** 0x30330244, 0x1, 0x00000000, 0x0, 0x303304AC
- #define **IOMUXC\_UART3\_RXD\_UART1\_RTS\_B** 0x30330244, 0x1, 0x303304F0, 0x0, 0x303304AC
- #define **IOMUXC\_UART3\_RXD\_GPIO5\_IO26** 0x30330244, 0x5, 0x00000000, 0x0, 0x303304AC
- #define **IOMUXC\_UART3\_TXD\_UART3\_TX** 0x30330248, 0x0, 0x00000000, 0x0, 0x303304B0
- #define **IOMUXC\_UART3\_TXD\_UART3\_RX** 0x30330248, 0x0, 0x30330504, 0x3, 0x303304B0
- #define **IOMUXC\_UART3\_TXD\_UART1\_RTS\_B** 0x30330248, 0x1, 0x303304F0, 0x1, 0x303304B0
- #define **IOMUXC\_UART3\_TXD\_UART1\_CTS\_B** 0x30330248, 0x1, 0x00000000, 0x0, 0x303304B0
- #define **IOMUXC\_UART3\_TXD\_GPIO5\_IO27** 0x30330248, 0x5, 0x00000000, 0x0, 0x303304B0
- #define **IOMUXC\_UART4\_RXD\_UART4\_RX** 0x3033024C, 0x0, 0x3033050C, 0x2, 0x303304B4
- #define **IOMUXC\_UART4\_RXD\_UART4\_TX** 0x3033024C, 0x0, 0x00000000, 0x0, 0x303304B4
- #define **IOMUXC\_UART4\_RXD\_UART2\_CTS\_B** 0x3033024C, 0x1, 0x00000000, 0x0, 0x303304B4
- #define **IOMUXC\_UART4\_RXD\_UART2\_RTS\_B** 0x3033024C, 0x1, 0x303304F8, 0x0, 0x303304B4
- #define **IOMUXC\_UART4\_RXD\_PCIE1\_CLKREQ\_B** 0x3033024C, 0x2, 0x30330524, 0x1, 0x303304B4
- #define **IOMUXC\_UART4\_RXD\_GPIO5\_IO28** 0x3033024C, 0x5, 0x00000000, 0x0, 0x303304B4
- #define **IOMUXC\_UART4\_TXD\_UART4\_TX** 0x30330250, 0x0, 0x00000000, 0x0, 0x303304B8
- #define **IOMUXC\_UART4\_TXD\_UART4\_RX** 0x30330250, 0x0, 0x3033050C, 0x3, 0x303304B8
- #define **IOMUXC\_UART4\_TXD\_UART2\_RTS\_B** 0x30330250, 0x1, 0x303304F8, 0x1,

## Function Documentation

- 0x303304B8
- #define **IOMUXC\_UART4\_TXD\_UART2\_CTS\_B** 0x30330250, 0x1, 0x00000000, 0x0, 0x303304B8
- #define **IOMUXC\_UART4\_TXD\_PCIE2\_CLKREQ\_B** 0x30330250, 0x2, 0x30330528, 0x1, 0x303304B8
- #define **IOMUXC\_UART4\_TXD\_GPIO5\_IO29** 0x30330250, 0x5, 0x00000000, 0x0, 0x303304B8
- #define **IOMUXC\_TEST\_MODE** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330254
- #define **IOMUXC\_BOOT\_MODE0** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330258
- #define **IOMUXC\_BOOT\_MODE1** 0x00000000, 0x0, 0x00000000, 0x0, 0x3033025C
- #define **IOMUXC\_JTAG\_MOD** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330260
- #define **IOMUXC\_JTAG\_TRST\_B** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330264
- #define **IOMUXC\_JTAG\_TDI** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330268
- #define **IOMUXC\_JTAG\_TMS** 0x00000000, 0x0, 0x00000000, 0x0, 0x3033026C
- #define **IOMUXC\_JTAG\_TCK** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330270
- #define **IOMUXC\_JTAG\_TDO** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330274
- #define **IOMUXC\_RTC** 0x00000000, 0x0, 0x00000000, 0x0, 0x30330278

## Configuration

- static void **IOMUXC\_SetPinMux** (uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t inputOnfield)  
*Sets the IOMUXC pin mux mode.*
- static void **IOMUXC\_SetPinConfig** (uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t configValue)  
*Sets the IOMUXC pin configuration.*

## 20.2 Macro Definition Documentation

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

## 20.3 Function Documentation

### 20.3.1 static void IOMUXC\_SetPinMux ( uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t inputOnfield ) [inline], [static]

Note

The first five parameters can be filled with the pin function ID macros.

This is an example to set the I2C4\_SDA as the pwm1\_OUT:

```
* IOMUXC_SetPinMux(IOMUXC_I2C4_SDA_PWM1_OUT, 0);  
*
```

## Parameters

<i>muxRegister</i>	The pin mux register_
<i>muxMode</i>	The pin mux mode_
<i>inputRegister</i>	The select input register_
<i>inputDaisy</i>	The input daisy_
<i>configRegister</i>	The config register_
<i>inputInv</i>	The pad->module input inversion_

**20.3.2 static void IOMUXC\_SetPinConfig ( uint32\_t *muxRegister*, uint32\_t *muxMode*, uint32\_t *inputRegister*, uint32\_t *inputDaisy*, uint32\_t *configRegister*, uint32\_t *configValue* ) [inline], [static]**

## Note

The previous five parameters can be filled with the pin function ID macros.

This is an example to set pin configuration for IOMUXC\_I2C4\_SDA\_PWM1\_OUT:

```
* IOMUXC_SetPinConfig(IOMUXC_I2C4_SDA_PWM1_OUT, IOMUXC_SW_PAD_CTL_PAD_ODE_MASK |
  IOMUXC0_SW_PAD_CTL_PAD_DSE(2U) )
*
```

## Parameters

<i>muxRegister</i>	The pin mux register_
<i>muxMode</i>	The pin mux mode_
<i>inputRegister</i>	The select input register_
<i>inputDaisy</i>	The input daisy_
<i>configRegister</i>	The config register_
<i>configValue</i>	The pin config value_



## Chapter 21 Debug Console

### 21.1 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.

### 21.2 Function groups

#### 21.2.1 Initialization

To initialize the debug console, call the [DbgConsole\\_Init\(\)](#) function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate, serial_port_type_t device,
                        uint32_t clkSrcFreq);
```

Selects the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_Uart = 1U,
    kSerialPort_UsbCdc,
    kSerialPort_Swo,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral.

This example shows how to call the [DbgConsole\\_Init\(\)](#) given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,
                BOARD_DEBUG_UART_CLK_FREQ);
```

#### 21.2.2 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

## Function groups

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

<b>.precision</b>	<b>Description</b>
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

<b>length</b>	<b>Description</b>
Do not support	

<b>specifier</b>	<b>Description</b>
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

## Function groups

- Support a format specifier for SCANF following this prototype " %[\*][width][length]specifier", which is explained below

*	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *



specifier	Qualifying Input	Type of argument
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(const char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE    /* Select printf, scanf, putchar, getchar of SDK version. */
#define PRINTF           DbgConsole_Printf
#define SCANF            DbgConsole_Scanf
#define PUTCHAR          DbgConsole_Putchar
#define GETCHAR          DbgConsole_Getchar
#else                   /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF           printf
#define SCANF            scanf
#define PUTCHAR          putchar
#define GETCHAR          getchar
#endif /* SDK_DEBUGCONSOLE */
```

## 21.3 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\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

### Print out failure messages using MCUXpresso SDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
    , line, func);
    for (;;)
    {}
}
```

### Note:

To use 'printf' and 'scanf' for GNUC Base, add file 'fsl\_sbrk.c' in path: ..\{package}\devices\{subset}\utilities\fsl-\_sbrk.c to your project.

## Modules

- [SWO](#)  
/\*!
- [Semihosting](#)

## Macros

- `#define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U`  
*Definition select redirect toolchain printf, scanf to uart or not.*
- `#define DEBUGCONSOLE_REDIRECT_TO_SDK 1U`  
*Select SDK version printf, scanf.*
- `#define DEBUGCONSOLE_DISABLE 2U`  
*Disable debugconsole function.*
- `#define SDK_DEBUGCONSOLE 1U`  
*Definition to select sdk or toolchain printf, scanf.*
- `#define SDK_DEBUGCONSOLE_UART`  
*Definition to select redirect toolchain printf, scanf to uart or not.*
- `#define PRINTF DbgConsole_Printf`  
*Definition to select redirect toolchain printf, scanf to uart or not.*

## Typedefs

- `typedef void(* printfCb )(char *buf, int32_t *indicator, char val, int len)`  
*A function pointer which is used when format printf log.*

## Functions

- `int StrFormatPrintf (const char *fmt, va_list ap, char *buf, printfCb cb)`  
*This function outputs its parameters according to a formatted string.*
- `int StrFormatScanf (const char *line_ptr, char *format, va_list args_ptr)`  
*Converts an input line of ASCII characters based upon a provided string format.*

## Initialization

- `status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)`  
*Initializes the peripheral used for debug messages.*
- `status_t DbgConsole_Deinit (void)`  
*De-initializes the peripheral used for debug messages.*
- `int DbgConsole_Printf (const char *formatString,...)`  
*Writes formatted output to the standard output stream.*
- `int DbgConsole_Putchar (int ch)`  
*Writes a character to stdout.*
- `int DbgConsole_Scanf (char *formatString,...)`  
*Reads formatted data from the standard input stream.*
- `int DbgConsole_Getchar (void)`  
*Reads a character from standard input.*
- `status_t DbgConsole_Flush (void)`  
*Debug console flush.*

## 21.4 Macro Definition Documentation

### 21.4.1 `#define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U`

Select toolchain printf and scanf.

## Function Documentation

**21.4.2 #define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U**

**21.4.3 #define DEBUGCONSOLE\_DISABLE 2U**

**21.4.4 #define SDK\_DEBUGCONSOLE 1U**

**21.4.5 #define SDK\_DEBUGCONSOLE\_UART**

**21.4.6 #define PRINTF DbgConsole\_Printf**

if SDK\_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK\_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK\_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

## 21.5 Function Documentation

**21.5.1 status\_t DbgConsole\_Init ( uint8\_t *instance*, uint32\_t *baudRate*, serial\_port\_type\_t *device*, uint32\_t *clkSrcFreq* )**

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

<i>instance</i>	The instance of the module.
<i>baudRate</i>	The desired baud rate in bits per second.
<i>device</i>	Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"><li>• kSerialPort_Uart,</li><li>• kSerialPort_UsbCdc.</li></ul>
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully
------------------------	------------------------

### 21.5.2 **status\_t DbgConsole\_Deinit ( void )**

Call this function to disable debug log messages to be output via the specified peripheral initialized by the serial manager module.

Returns

Indicates whether de-initialization was successful or not.

### 21.5.3 **int DbgConsole\_Printf ( const char \* *formatString*, ... )**

Call this function to write a formatted output to the standard output stream.

Parameters

<i>formatString</i>	Format control string.
---------------------	------------------------

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 21.5.4 **int DbgConsole\_Putchar ( int *ch* )**

Call this function to write a character to stdout.

Parameters

<i>ch</i>	Character to be written.
-----------	--------------------------

Returns

Returns the character written.

## Function Documentation

### 21.5.5 int DbgConsole\_Scanf ( char \* *formatString*, ... )

Call this function to read formatted data from the standard input stream.

#### Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function `DbgConsole_TryGetchar` to get the input char.

#### Parameters

<i>formatString</i>	Format control string.
---------------------	------------------------

#### Returns

Returns the number of fields successfully converted and assigned.

### 21.5.6 int DbgConsole\_Getchar ( void )

Call this function to read a character from standard input.

#### Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function `DbgConsole_TryGetchar` to get the input char.

#### Returns

Returns the character read.

### 21.5.7 status\_t DbgConsole\_Flush ( void )

Call this function to wait the tx buffer empty. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

#### Returns

Indicates whether wait idle was successful or not.

### 21.5.8 int StrFormatPrintf ( const char \* *fmt*, va\_list *ap*, char \* *buf*, printfCb *cb* )

Note

I/O is performed by calling given function pointer using following (\*func\_ptr)(c);

Parameters

in	<i>fmt</i>	Format string for printf.
in	<i>ap</i>	Arguments to printf.
in	<i>buf</i>	pointer to the buffer
	<i>cb</i>	print callbck function pointer

Returns

Number of characters to be print

### 21.5.9 int StrFormatScanf ( const char \* *line\_ptr*, char \* *format*, va\_list *args\_ptr* )

Parameters

in	<i>line_ptr</i>	The input line of ASCII data.
in	<i>format</i>	Format first points to the format string.
in	<i>args_ptr</i>	The list of parameters.

Returns

Number of input items converted and assigned.

Return values

<i>IO_EOF</i>	When line_ptr is empty string "".
---------------	-----------------------------------

### 21.6 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

#### 21.6.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the `SDK_DEBUGCONSOLE` is disabled.

##### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

##### Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

##### Step 3: Starting semihosting

1. Choose "Semihosting\_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
  2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
  3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting.
1. Make sure the `SDK_DEBUGCONSOLE_UART` is not defined, remove the default definition in `fsl_debug_console.h`.
  1. Start the project by choosing Project>Download and Debug.
  2. Choose View>Terminal I/O to display the output from the I/O operations.

#### 21.6.2 Guide Semihosting for Keil $\mu$ Vision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.



### 21.6.3 Guide Semihosting for MCUXpresso IDE

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK\_DEBUGCONSOLE=0, if set SDK\_DEBUGCONSOLE=1, the log will be redirect to the UART.

#### Step 2: Building the project

1. Compile and link the project.

#### Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

### 21.6.4 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__heap_size__=0x2000")
```

### Step 2: Building the project

1. Change "CMakeLists.txt":

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"

**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

**Replace paragraph**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-common")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffunction-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fdata-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffreestanding")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-builtin")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mthumb")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mapcs")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --gc-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -static")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -z")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} muldefs")

**To**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --specs=rdimon.specs ")

**Remove**

target\_link\_libraries(semihosting\_ARMGCC.elf debug nosys)

2. Run "build\_debug.bat" to build project

### Step 3: Starting semihosting

- (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.

## SWO

### 21.7 SWO

/\*!

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDE support SWO also, such IAR and KEIL, both input and output are supported, reference below for detail.

#### 21.7.1 Guide SWO for SDK

**NOTE:** After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

##### Step 1: Setting up the environment

1. Define `DEBUG_CONSOLE_IO_SWO` in your project settings.
2. Prepare code, the port and baudrate can be decided by application, `clkSrcFreq` should be mcu core clock frequency:

```
DbgConsole_Init(port, baudrate, DEBUG_CONSOLE_DEVICE_TYPE_SWO, clkSrcFreq);
```

3. Use `PRINTF` or `printf` to print some thing in application.

##### Step 2: Building the project

##### Step 3: Download and run project

#### 21.7.1.1 Guide SWO for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

##### Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK, and click the Activate button to enable trace data collection.

6. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log, The SDK\_DEBUGCONSOLE\_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero, then debug function ok. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one, then debug function ok.

**NOTE:** Case a or c only apply at example which enable swo function, the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h. For case a and c, Do and not do the above third step will be not affect function.

1. Start the project by choosing Project>Download and Debug.

### Step 2: Building the project

### Step 3: Starting swo

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.
3. Run application.

## 21.7.2 Guide SWO for Keil $\mu$ Vision

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

### Step 1: Setting up the environment

1. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log, the SDK\_DEBUGCONSOLE\_UART definition does not affect the functionality and skip the second step directly. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero, then start the second step. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one, then skip the second step directly.

**NOTE:** Case a or c only apply at example which enable swo function, the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h.

1. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
2. Open Project>Options for target or using Alt+F7 or click.
3. Select "Debug" tab, select "J-Link/J-Trace Cortex" and click "Setting button".
4. Select "Debug" tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click OK, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

### Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

## SWO

### Step 4: Run the project

1. Choose “Debug” on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

### 21.7.3 Guide SWO for MCUXpresso IDE

**NOTE:** MCUX support SWO for LPC-Link2 debug probe only.

### 21.7.4 Guide SWO for ARMGCC

**NOTE:** ARMGCC has no library support SWO.

**How to Reach Us:****Home Page:**[nxp.com](http://nxp.com)**Web Support:**[nxp.com/support](http://nxp.com/support)

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

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.

