
Document Number: MCUXSDKAPIRM
Rev 2.14.0
Jul 2023

MCUXpresso SDK API Reference Manual

NXP Semiconductors



Contents

Chapter 1 Introduction

Chapter 2 Trademarks

Chapter 3 Architectural Overview

Chapter 4 Clock Driver

4.1	Overview	7
4.2	Data Structure Documentation	23
4.2.1	struct ccm_analog_frac_pll_config_t	23
4.2.2	struct ccm_analog_integer_pll_config_t	23
4.3	Macro Definition Documentation	24
4.3.1	FSL_CLOCK_DRIVER_VERSION	24
4.3.2	ECSPI_CLOCKS	24
4.3.3	ENET_CLOCKS	24
4.3.4	GPIO_CLOCKS	24
4.3.5	GPT_CLOCKS	24
4.3.6	I2C_CLOCKS	25
4.3.7	IOMUX_CLOCKS	25
4.3.8	IPMUX_CLOCKS	25
4.3.9	PWM_CLOCKS	25
4.3.10	RDC_CLOCKS	25
4.3.11	SAI_CLOCKS	26
4.3.12	RDC_SEMA42_CLOCKS	26
4.3.13	UART_CLOCKS	26
4.3.14	USDHC_CLOCKS	26
4.3.15	WDOG_CLOCKS	26
4.3.16	TMU_CLOCKS	27
4.3.17	SDMA_CLOCKS	27
4.3.18	MU_CLOCKS	27
4.3.19	QSPI_CLOCKS	27
4.3.20	PDM_CLOCKS	27
4.3.21	kCLOCK_CoreSysClk	28
4.3.22	CLOCK_GetCoreSysClkFreq	28

Section No.	Title	Page No.
4.4	Enumeration Type Documentation	28
4.4.1	clock_name_t	28
4.4.2	clock_ip_name_t	29
4.4.3	clock_root_control_t	30
4.4.4	clock_root_t	31
4.4.5	clock_rootmux_m4_clk_sel_t	32
4.4.6	clock_rootmux_axi_clk_sel_t	33
4.4.7	clock_rootmux_ahb_clk_sel_t	33
4.4.8	clock_rootmux_audio_ahb_clk_sel_t	33
4.4.9	clock_rootmux_qspi_clk_sel_t	34
4.4.10	clock_rootmux_ecspi_clk_sel_t	34
4.4.11	clock_rootmux_enet_axi_clk_sel_t	34
4.4.12	clock_rootmux_enet_ref_clk_sel_t	35
4.4.13	clock_rootmux_enet_timer_clk_sel_t	35
4.4.14	clock_rootmux_enet_phy_clk_sel_t	35
4.4.15	clock_rootmux_i2c_clk_sel_t	36
4.4.16	clock_rootmux_uart_clk_sel_t	36
4.4.17	clock_rootmux_gpt_t	36
4.4.18	clock_rootmux_wdog_clk_sel_t	37
4.4.19	clock_rootmux_Pwm_clk_sel_t	37
4.4.20	clock_rootmux_sai_clk_sel_t	37
4.4.21	clock_rootmux_pdm_clk_sel_t	38
4.4.22	clock_rootmux_noc_clk_sel_t	38
4.4.23	clock_pll_gate_t	38
4.4.24	clock_gate_value_t	39
4.4.25	clock_pll_bypass_ctrl_t	39
4.4.26	clock_pll_clke_t	40
4.4.27	anonymous enum	40
4.5	Function Documentation	41
4.5.1	CLOCK_SetRootMux	41
4.5.2	CLOCK_GetRootMux	42
4.5.3	CLOCK_EnableRoot	42
4.5.4	CLOCK_DisableRoot	42
4.5.5	CLOCK_IsRootEnabled	42
4.5.6	CLOCK_UpdateRoot	43
4.5.7	CLOCK_SetRootDivider	43
4.5.8	CLOCK_GetRootPreDivider	43
4.5.9	CLOCK_GetRootPostDivider	44
4.5.10	CLOCK_ControlGate	44
4.5.11	CLOCK_EnableClock	44
4.5.12	CLOCK_DisableClock	45
4.5.13	CLOCK_PowerUpPll	45
4.5.14	CLOCK_PowerDownPll	45
4.5.15	CLOCK_SetPllBypass	45

Section No.	Title	Page No.
4.5.16	CLOCK_IsPllBypassed	46
4.5.17	CLOCK_IsPllLocked	46
4.5.18	CLOCK_EnableAnalogClock	46
4.5.19	CLOCK_DisableAnalogClock	47
4.5.20	CLOCK_OverridePllClke	47
4.5.21	CLOCK_OverridePllPd	47
4.5.22	CLOCK_InitArmPll	48
4.5.23	CLOCK_InitSysPll1	48
4.5.24	CLOCK_InitSysPll2	48
4.5.25	CLOCK_InitSysPll3	48
4.5.26	CLOCK_InitAudioPll1	49
4.5.27	CLOCK_InitAudioPll2	49
4.5.28	CLOCK_InitVideoPll1	49
4.5.29	CLOCK_InitIntegerPll	50
4.5.30	CLOCK_GetIntegerPllFreq	50
4.5.31	CLOCK_InitFracPll	50
4.5.32	CLOCK_GetFracPllFreq	51
4.5.33	CLOCK_GetPllFreq	51
4.5.34	CLOCK_GetPllRefClkFreq	51
4.5.35	CLOCK_GetFreq	52
4.5.36	CLOCK_GetClockRootFreq	52
4.5.37	CLOCK_GetCoreM4Freq	52
4.5.38	CLOCK_GetAxiFreq	53
4.5.39	CLOCK_GetAhbFreq	53
4.5.40	CLOCK_GetEnetAxiFreq	53

Chapter 5 IOMUXC: IOMUX Controller

5.1	Overview	54
5.2	Macro Definition Documentation	73
5.2.1	FSL_IOMUXC_DRIVER_VERSION	73
5.3	Function Documentation	73
5.3.1	IOMUXC_SetPinMux	73
5.3.2	IOMUXC_SetPinConfig	74

Chapter 6 Common Driver

6.1	Overview	75
6.2	Macro Definition Documentation	79
6.2.1	FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ	79
6.2.2	MAKE_STATUS	79
6.2.3	MAKE_VERSION	79

Section No.	Title	Page No.
6.2.4	FSL_COMMON_DRIVER_VERSION	80
6.2.5	DEBUG_CONSOLE_DEVICE_TYPE_NONE	80
6.2.6	DEBUG_CONSOLE_DEVICE_TYPE_UART	80
6.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	80
6.2.8	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	80
6.2.9	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	80
6.2.10	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	80
6.2.11	DEBUG_CONSOLE_DEVICE_TYPE_IUART	80
6.2.12	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	80
6.2.13	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	80
6.2.14	DEBUG_CONSOLE_DEVICE_TYPE_SWO	80
6.2.15	DEBUG_CONSOLE_DEVICE_TYPE_QSCI	80
6.2.16	ARRAY_SIZE	80
6.2.17	SDK_SIZEALIGN	80
6.3	Typedef Documentation	80
6.3.1	status_t	80
6.4	Enumeration Type Documentation	81
6.4.1	_status_groups	81
6.4.2	anonymous enum	83
6.5	Function Documentation	84
6.5.1	SDK_Malloc	84
6.5.2	SDK_Free	84
6.5.3	SDK_DelayAtLeastUs	84
6.5.4	EnableIRQ	84
6.5.5	DisableIRQ	85
6.5.6	EnableIRQWithPriority	85
6.5.7	IRQ_SetPriority	86
6.5.8	IRQ_ClearPendingIRQ	86
6.5.9	DisableGlobalIRQ	87
6.5.10	EnableGlobalIRQ	87
 Chapter 7 ECSPI: Enhanced Configurable Serial Peripheral Interface Driver		
7.1	Overview	88
7.2	ECSPI Driver	89
7.2.1	Overview	89
7.2.2	Typical use case	89
7.2.3	Data Structure Documentation	94
7.2.4	Macro Definition Documentation	97
7.2.5	Enumeration Type Documentation	97
7.2.6	Function Documentation	99

Section No.	Title	Page No.
7.3	ECSPI FreeRTOS Driver	112
7.3.1	Overview	112
7.3.2	Macro Definition Documentation	112
7.3.3	Function Documentation	112
7.4	ECSPI SDMA Driver	115
7.4.1	Overview	115
7.4.2	Data Structure Documentation	116
7.4.3	Macro Definition Documentation	116
7.4.4	Typedef Documentation	116
7.4.5	Function Documentation	116
7.5	ECSPI CMSIS Driver	120
7.5.1	Function groups	120
7.5.2	Typical use case	121
 Chapter 8 ENET: Ethernet MAC Driver		
8.1	Overview	122
8.2	Operations of Ethernet MAC Driver	122
8.2.1	MII interface Operation	122
8.2.2	MAC address filter	122
8.2.3	Other Basic control Operations	122
8.2.4	Transactional Operation	122
8.2.5	PTP IEEE 1588 Feature Operation	123
8.3	Typical use case	123
8.3.1	ENET Initialization, receive, and transmit operations	123
8.4	Data Structure Documentation	130
8.4.1	struct enet_rx_bd_struct_t	130
8.4.2	struct enet_tx_bd_struct_t	131
8.4.3	struct enet_data_error_stats_t	131
8.4.4	struct enet_rx_frame_error_t	132
8.4.5	struct enet_transfer_stats_t	132
8.4.6	struct enet_frame_info_t	133
8.4.7	struct enet_tx_dirty_ring_t	133
8.4.8	struct enet_buffer_config_t	134
8.4.9	struct enet_config_t	135
8.4.10	struct enet_tx_bd_ring_t	138
8.4.11	struct enet_rx_bd_ring_t	138
8.4.12	struct _enet_handle	139
8.5	Macro Definition Documentation	141
8.5.1	FSL_ENET_DRIVER_VERSION	141

Section No.	Title	Page No.
8.5.2	FSL_FEATURE_ENET_QUEUE	141
8.5.3	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	141
8.5.4	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	141
8.5.5	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	141
8.5.6	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	141
8.5.7	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	141
8.5.8	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	141
8.5.9	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	141
8.5.10	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	141
8.5.11	ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK	141
8.5.12	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	141
8.5.13	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	141
8.5.14	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	141
8.5.15	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	141
8.5.16	ENET_BUFFDESCRIPTOR_TX_READY_MASK	141
8.5.17	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK	141
8.5.18	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	141
8.5.19	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK	141
8.5.20	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	141
8.5.21	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	141
8.5.22	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	141
8.5.23	ENET_FRAME_MAX_FRAMELEN	142
8.5.24	ENET_FRAME_VLAN_TAGLEN	142
8.5.25	ENET_FRAME_CRC_LEN	142
8.5.26	ENET_FIFO_MIN_RX_FULL	142
8.5.27	ENET_RX_MIN_BUFFERSIZE	142
8.5.28	ENET_PHY_MAXADDRESS	142
8.5.29	ENET_TX_INTERRUPT	142
8.5.30	ENET_RX_INTERRUPT	142
8.5.31	ENET_TS_INTERRUPT	142
8.5.32	ENET_ERR_INTERRUPT	142
8.6	Typedef Documentation	143
8.6.1	enet_rx_alloc_callback_t	143
8.6.2	enet_rx_free_callback_t	143
8.6.3	enet_callback_t	143
8.6.4	enet_isr_t	143
8.7	Enumeration Type Documentation	143
8.7.1	anonymous enum	143
8.7.2	enet_mii_mode_t	143
8.7.3	enet_mii_speed_t	143
8.7.4	enet_mii_duplex_t	144
8.7.5	enet_mii_write_t	144
8.7.6	enet_mii_read_t	144

Section No.	Title	Page No.
8.7.7	enet_special_control_flag_t	144
8.7.8	enet_interrupt_enable_t	145
8.7.9	enet_event_t	145
8.7.10	enet_tx_accelerator_t	145
8.7.11	enet_rx_accelerator_t	146
8.8	Function Documentation	146
8.8.1	ENET_GetInstance	146
8.8.2	ENET_GetDefaultConfig	146
8.8.3	ENET_Up	146
8.8.4	ENET_Init	147
8.8.5	ENET_Down	148
8.8.6	ENET_Deinit	148
8.8.7	ENET_Reset	149
8.8.8	ENET_SetMII	149
8.8.9	ENET_SetSMI	149
8.8.10	ENET_GetSMI	150
8.8.11	ENET_ReadSMIData	151
8.8.12	ENET_StartSMIWrite	151
8.8.13	ENET_StartSMIRead	152
8.8.14	ENET_MDIOWrite	152
8.8.15	ENET_MDIORead	152
8.8.16	ENET_SetMacAddr	153
8.8.17	ENET_GetMacAddr	153
8.8.18	ENET_AddMulticastGroup	153
8.8.19	ENET_LeaveMulticastGroup	154
8.8.20	ENET_ActiveRead	154
8.8.21	ENET_EnableSleepMode	154
8.8.22	ENET_GetAccelFunction	155
8.8.23	ENET_EnableInterrupts	155
8.8.24	ENET_DisableInterrupts	155
8.8.25	ENET_GetInterruptStatus	156
8.8.26	ENET_ClearInterruptStatus	156
8.8.27	ENET_SetRxISRHandler	156
8.8.28	ENET_SetTxISRHandler	157
8.8.29	ENET_SetErrISRHandler	157
8.8.30	ENET_GetRxErrBeforeReadFrame	157
8.8.31	ENET_GetStatistics	158
8.8.32	ENET_GetRxFrameSize	158
8.8.33	ENET_ReadFrame	159
8.8.34	ENET_SendFrame	160
8.8.35	ENET_SetTxReclaim	161
8.8.36	ENET_ReclaimTxDescriptor	161
8.8.37	ENET_GetRxFrame	162
8.8.38	ENET_StartTxFrame	163

Section No.		Title	Page No.
8.8.39		ENET_TransmitIRQHandler	163
8.8.40		ENET_ReceiveIRQHandler	164
8.8.41		ENET_ErrorIRQHandler	164
8.8.42		ENET_Ptp1588IRQHandler	164
8.8.43		ENET_CommonFrame0IRQHandler	164
8.9	Variable Documentation		165
8.9.1		s_enetClock	165
Chapter 9 GPC: General Power Controller Driver			
9.1	Overview		166
9.2	Macro Definition Documentation		167
9.2.1		FSL_GPC_DRIVER_VERSION	167
9.3	Enumeration Type Documentation		168
9.3.1		_gpc_lpm_mode	168
9.3.2		_gpc_pgc_ack_sel	168
9.3.3		_gpc_standby_count	168
9.4	Function Documentation		168
9.4.1		GPC_AllowIRQs	168
9.4.2		GPC_DisallowIRQs	169
9.4.3		GPC_GetLpmMode	169
9.4.4		GPC_EnableIRQ	169
9.4.5		GPC_DisableIRQ	169
9.4.6		GPC_GetIRQStatusFlag	170
9.4.7		GPC_DsmTriggerMask	170
9.4.8		GPC_WFIMask	170
9.4.9		GPC_SelectPGCAckSignal	170
9.4.10		GPC_PowerDownRequestMask	171
9.4.11		GPC_PGCMapping	171
9.4.12		GPC_TimeSlotConfigureForPUS	171
9.4.13		GPC_EnterWaitMode	171
9.4.14		GPC_EnterStopMode	172
9.4.15		GPC_Init	172
Chapter 10 GPT: General Purpose Timer			
10.1	Overview		173
10.2	Function groups		173
10.2.1		Initialization and deinitialization	173
10.3	Typical use case		173

Section No.	Title	Page No.
10.3.1	GPT interrupt example	173
10.4	Data Structure Documentation	176
10.4.1	struct gpt_config_t	176
10.5	Enumeration Type Documentation	177
10.5.1	gpt_clock_source_t	177
10.5.2	gpt_input_capture_channel_t	177
10.5.3	gpt_input_operation_mode_t	178
10.5.4	gpt_output_compare_channel_t	178
10.5.5	gpt_output_operation_mode_t	178
10.5.6	gpt_interrupt_enable_t	178
10.5.7	gpt_status_flag_t	179
10.6	Function Documentation	179
10.6.1	GPT_Init	179
10.6.2	GPT_Deinit	179
10.6.3	GPT_GetDefaultConfig	179
10.6.4	GPT_SoftwareReset	180
10.6.5	GPT_SetClockSource	180
10.6.6	GPT_GetClockSource	180
10.6.7	GPT_SetClockDivider	180
10.6.8	GPT_GetClockDivider	181
10.6.9	GPT_SetOscClockDivider	181
10.6.10	GPT_GetOscClockDivider	181
10.6.11	GPT_StartTimer	181
10.6.12	GPT_StopTimer	182
10.6.13	GPT_GetCurrentTimerCount	182
10.6.14	GPT_SetInputOperationMode	182
10.6.15	GPT_GetInputOperationMode	182
10.6.16	GPT_GetInputCaptureValue	183
10.6.17	GPT_SetOutputOperationMode	183
10.6.18	GPT_GetOutputOperationMode	184
10.6.19	GPT_SetOutputCompareValue	184
10.6.20	GPT_GetOutputCompareValue	184
10.6.21	GPT_ForceOutput	185
10.6.22	GPT_EnableInterrupts	185
10.6.23	GPT_DisableInterrupts	185
10.6.24	GPT_GetEnabledInterrupts	185
10.6.25	GPT_GetStatusFlags	186
10.6.26	GPT_ClearStatusFlags	186
Chapter 11	GPIO: General-Purpose Input/Output Driver	
11.1	Overview	187

Section No.	Title	Page No.
11.2	Typical use case	187
11.2.1	Input Operation	187
11.3	Data Structure Documentation	189
11.3.1	struct gpio_pin_config_t	189
11.4	Macro Definition Documentation	189
11.4.1	FSL_GPIO_DRIVER_VERSION	189
11.5	Enumeration Type Documentation	189
11.5.1	gpio_pin_direction_t	189
11.5.2	gpio_interrupt_mode_t	189
11.6	Function Documentation	190
11.6.1	GPIO_PinInit	190
11.6.2	GPIO_PinWrite	191
11.6.3	GPIO_WritePinOutput	191
11.6.4	GPIO_PortSet	191
11.6.5	GPIO_SetPinsOutput	191
11.6.6	GPIO_PortClear	192
11.6.7	GPIO_ClearPinsOutput	193
11.6.8	GPIO_PortToggle	193
11.6.9	GPIO_PinRead	193
11.6.10	GPIO_ReadPinInput	193
11.6.11	GPIO_PinReadPadStatus	194
11.6.12	GPIO_ReadPadStatus	195
11.6.13	GPIO_PinSetInterruptConfig	195
11.6.14	GPIO_SetPinInterruptConfig	195
11.6.15	GPIO_PortEnableInterrupts	195
11.6.16	GPIO_EnableInterrupts	196
11.6.17	GPIO_PortDisableInterrupts	196
11.6.18	GPIO_DisableInterrupts	196
11.6.19	GPIO_PortGetInterruptFlags	196
11.6.20	GPIO_GetPinsInterruptFlags	197
11.6.21	GPIO_PortClearInterruptFlags	197
11.6.22	GPIO_ClearPinsInterruptFlags	197
Chapter 12 I2C: Inter-Integrated Circuit Driver		
12.1	Overview	199
12.2	I2C Driver	200
12.2.1	Overview	200
12.2.2	Typical use case	200
12.2.3	Data Structure Documentation	204

Section No.	Title	Page No.
12.2.4	Macro Definition Documentation	208
12.2.5	Typedef Documentation	208
12.2.6	Enumeration Type Documentation	208
12.2.7	Function Documentation	210
12.3	I2C FreeRTOS Driver	223
12.3.1	Overview	223
12.3.2	Macro Definition Documentation	223
12.3.3	Function Documentation	223
12.4	I2C CMSIS Driver	226
12.4.1	I2C CMSIS Driver	226
Chapter 13 PWM: Pulse Width Modulation Driver		
13.1	Overview	228
13.2	PWM Driver	228
13.2.1	Initialization and deinitialization	228
13.3	Typical use case	228
13.3.1	PWM output	228
13.4	Enumeration Type Documentation	230
13.4.1	pwm_clock_source_t	230
13.4.2	pwm_fifo_water_mark_t	231
13.4.3	pwm_byte_data_swap_t	231
13.4.4	pwm_half_word_data_swap_t	231
13.4.5	pwm_output_configuration_t	231
13.4.6	pwm_sample_repeat_t	232
13.4.7	pwm_interrupt_enable_t	232
13.4.8	pwm_status_flags_t	232
13.4.9	pwm_fifo_available_t	232
13.5	Function Documentation	233
13.5.1	PWM_Init	233
13.5.2	PWM_Deinit	233
13.5.3	PWM_GetDefaultConfig	233
13.5.4	PWM_StartTimer	234
13.5.5	PWM_StopTimer	234
13.5.6	PWM_SoftwareReset	234
13.5.7	PWM_EnableInterrupts	234
13.5.8	PWM_DisableInterrupts	235
13.5.9	PWM_GetEnabledInterrupts	235
13.5.10	PWM_GetStatusFlags	235
13.5.11	PWM_clearStatusFlags	236

Section No.	Title	Page No.
13.5.12	PWM_GetFIFOAvailable	237
13.5.13	PWM_SetSampleValue	237
13.5.14	PWM_GetSampleValue	237
13.5.15	PWM_SetPeriodValue	238
13.5.16	PWM_GetPeriodValue	239
13.5.17	PWM_GetCounterValue	239
 Chapter 14 UART: Universal Asynchronous Receiver/Transmitter Driver		
14.1	Overview	240
14.2	UART Driver	241
14.2.1	Overview	241
14.2.2	Typical use case	241
14.2.3	Data Structure Documentation	247
14.2.4	Macro Definition Documentation	250
14.2.5	Typedef Documentation	250
14.2.6	Enumeration Type Documentation	250
14.2.7	Function Documentation	253
14.2.8	Variable Documentation	266
14.3	UART FreeRTOS Driver	267
14.3.1	Overview	267
14.3.2	Data Structure Documentation	267
14.3.3	Macro Definition Documentation	268
14.3.4	Function Documentation	268
14.4	UART SDMA Driver	270
14.4.1	Overview	270
14.4.2	Data Structure Documentation	271
14.4.3	Macro Definition Documentation	272
14.4.4	Typedef Documentation	272
14.4.5	Function Documentation	272
14.5	UART CMSIS Driver	276
14.5.1	Function groups	276
 Chapter 15 MU: Messaging Unit		
15.1	Overview	278
15.2	Function description	278
15.2.1	MU initialization	278
15.2.2	MU message	278
15.2.3	MU flags	279

Section No.	Title	Page No.
15.2.4	Status and interrupt	279
15.2.5	MU misc functions	279
15.3	Macro Definition Documentation	282
15.3.1	FSL_MU_DRIVER_VERSION	282
15.4	Enumeration Type Documentation	282
15.4.1	_mu_status_flags	282
15.4.2	_mu_interrupt_enable	282
15.4.3	_mu_interrupt_trigger	283
15.5	Function Documentation	283
15.5.1	MU_Init	283
15.5.2	MU_Deinit	283
15.5.3	MU_SendMsgNonBlocking	283
15.5.4	MU_SendMsg	284
15.5.5	MU_ReceiveMsgNonBlocking	284
15.5.6	MU_ReceiveMsg	285
15.5.7	MU_SetFlagsNonBlocking	285
15.5.8	MU_SetFlags	286
15.5.9	MU_GetFlags	286
15.5.10	MU_GetStatusFlags	286
15.5.11	MU_GetInterruptsPending	287
15.5.12	MU_ClearStatusFlags	287
15.5.13	MU_EnableInterrupts	288
15.5.14	MU_DisableInterrupts	288
15.5.15	MU_TriggerInterrupts	289
15.5.16	MU_ClearNmi	289
15.5.17	MU_BootCoreB	289
15.5.18	MU_HoldCoreBReset	290
15.5.19	MU_BootOtherCore	290
15.5.20	MU_HoldOtherCoreReset	290
15.5.21	MU_ResetBothSides	291
15.5.22	MU_HardwareResetOtherCore	291
15.5.23	MU_SetClockOnOtherCoreEnable	292
15.5.24	MU_GetOtherCorePowerMode	292
 Chapter 16 PDM: Microphone Interface		
16.1	Overview	293
16.2	Typical use case	293
16.3	PDM Driver	294
16.3.1	Overview	294

Section No.	Title	Page No.
16.3.2	Typical use case	294
16.3.3	Data Structure Documentation	302
16.3.4	Enumeration Type Documentation	305
16.3.5	Function Documentation	309
16.4	PDM SDMA Driver	326
16.4.1	Typical use case	326
16.4.2	Overview	326
16.4.3	Data Structure Documentation	327
16.4.4	Function Documentation	328
Chapter 17 RDC: Resource Domain Controller		
17.1	Overview	331
17.2	Data Structure Documentation	333
17.2.1	struct rdc_hardware_config_t	333
17.2.2	struct rdc_domain_assignment_t	333
17.2.3	struct rdc_periph_access_config_t	333
17.2.4	struct rdc_mem_access_config_t	334
17.2.5	struct rdc_mem_status_t	335
17.3	Enumeration Type Documentation	335
17.3.1	_rdc_interrupts	335
17.3.2	_rdc_flags	335
17.3.3	_rdc_access_policy	336
17.4	Function Documentation	336
17.4.1	RDC_Init	336
17.4.2	RDC_Deinit	336
17.4.3	RDC_GetHardwareConfig	336
17.4.4	RDC_EnableInterrupts	336
17.4.5	RDC_DisableInterrupts	337
17.4.6	RDC_GetInterruptStatus	337
17.4.7	RDC_ClearInterruptStatus	337
17.4.8	RDC_GetStatus	337
17.4.9	RDC_ClearStatus	338
17.4.10	RDC_SetMasterDomainAssignment	338
17.4.11	RDC_GetDefaultMasterDomainAssignment	338
17.4.12	RDC_LockMasterDomainAssignment	339
17.4.13	RDC_SetPeriphAccessConfig	339
17.4.14	RDC_GetDefaultPeriphAccessConfig	339
17.4.15	RDC_LockPeriphAccessConfig	340
17.4.16	RDC_GetPeriphAccessPolicy	340
17.4.17	RDC_SetMemAccessConfig	340

Section No.	Title	Page No.
17.4.18	RDC_GetDefaultMemAccessConfig	341
17.4.19	RDC_LockMemAccessConfig	341
17.4.20	RDC_SetMemAccessValid	341
17.4.21	RDC_GetMemViolationStatus	342
17.4.22	RDC_ClearMemViolationFlag	342
17.4.23	RDC_GetMemAccessPolicy	342
17.4.24	RDC_GetCurrentMasterDomainId	342
 Chapter 18 RDC_SEMA42: Hardware Semaphores Driver		
18.1	Overview	344
18.2	Macro Definition Documentation	345
18.2.1	RDC_SEMA42_GATE_NUM_RESET_ALL	345
18.2.2	RDC_SEMA42_GATEn	345
18.2.3	RDC_SEMA42_GATE_COUNT	345
18.3	Function Documentation	345
18.3.1	RDC_SEMA42_Init	345
18.3.2	RDC_SEMA42_Deinit	345
18.3.3	RDC_SEMA42_TryLock	346
18.3.4	RDC_SEMA42_Lock	346
18.3.5	RDC_SEMA42_Unlock	347
18.3.6	RDC_SEMA42_GetLockMasterIndex	347
18.3.7	RDC_SEMA42_GetLockDomainID	347
18.3.8	RDC_SEMA42_ResetGate	348
18.3.9	RDC_SEMA42_ResetAllGates	349
 Chapter 19 SAI: Serial Audio Interface		
19.1	Overview	350
19.2	Typical configurations	350
19.3	Typical use case	351
19.3.1	SAI Send/receive using an interrupt method	351
19.3.2	SAI Send/receive using a DMA method	351
19.4	Typical use case	351
19.5	SAI Driver	352
19.5.1	Overview	352
19.5.2	Data Structure Documentation	359
19.5.3	Macro Definition Documentation	362
19.5.4	Enumeration Type Documentation	362
19.5.5	Function Documentation	366

Section No.	Title	Page No.
19.6	SAI SDMA Driver	398
19.6.1	Typical use case	398
19.6.2	Overview	398
19.6.3	Data Structure Documentation	399
19.6.4	Function Documentation	400
Chapter 20 SDMA: Smart Direct Memory Access (SDMA) Controller Driver		
20.1	Overview	406
20.2	Typical use case	406
20.2.1	SDMA Operation	406
20.3	Data Structure Documentation	412
20.3.1	struct sdma_config_t	412
20.3.2	struct sdma_multi_fifo_config_t	412
20.3.3	struct sdma_sw_done_config_t	412
20.3.4	struct sdma_p2p_config_t	413
20.3.5	struct sdma_transfer_config_t	413
20.3.6	struct sdma_buffer_descriptor_t	414
20.3.7	struct sdma_channel_control_t	415
20.3.8	struct sdma_context_data_t	415
20.3.9	struct sdma_handle_t	415
20.4	Macro Definition Documentation	416
20.4.1	FSL_SDMA_DRIVER_VERSION	416
20.5	Typedef Documentation	416
20.5.1	sdma_callback	416
20.6	Enumeration Type Documentation	416
20.6.1	sdma_transfer_size_t	416
20.6.2	sdma_bd_status_t	417
20.6.3	sdma_bd_command_t	417
20.6.4	sdma_context_switch_mode_t	417
20.6.5	sdma_clock_ratio_t	418
20.6.6	sdma_transfer_type_t	418
20.6.7	sdma_peripheral_t	418
20.6.8	anonymous enum	418
20.6.9	anonymous enum	419
20.6.10	anonymous enum	419
20.6.11	anonymous enum	419
20.6.12	sdma_done_src_t	419
20.7	Function Documentation	420
20.7.1	SDMA_Init	420

Section No.	Title	Page No.
20.7.2	SDMA_Deinit	421
20.7.3	SDMA_GetDefaultConfig	422
20.7.4	SDMA_ResetModule	422
20.7.5	SDMA_EnableChannelErrorInterrupts	422
20.7.6	SDMA_DisableChannelErrorInterrupts	422
20.7.7	SDMA_ConfigBufferDescriptor	423
20.7.8	SDMA_SetChannelPriority	423
20.7.9	SDMA_SetSourceChannel	424
20.7.10	SDMA_StartChannelSoftware	424
20.7.11	SDMA_StartChannelEvents	424
20.7.12	SDMA_StopChannel	424
20.7.13	SDMA_SetContextSwitchMode	425
20.7.14	SDMA_GetChannelInterruptStatus	425
20.7.15	SDMA_ClearChannelInterruptStatus	425
20.7.16	SDMA_GetChannelStopStatus	425
20.7.17	SDMA_ClearChannelStopStatus	426
20.7.18	SDMA_GetChannelPendStatus	426
20.7.19	SDMA_ClearChannelPendStatus	426
20.7.20	SDMA_GetErrorStatus	427
20.7.21	SDMA_GetRequestSourceStatus	427
20.7.22	SDMA_CreateHandle	427
20.7.23	SDMA_InstallBDMemory	428
20.7.24	SDMA_SetCallback	429
20.7.25	SDMA_SetMultiFifoConfig	429
20.7.26	SDMA_EnableSwDone	429
20.7.27	SDMA_SetDoneConfig	430
20.7.28	SDMA_LoadScript	430
20.7.29	SDMA_DumpScript	430
20.7.30	SDMA_GetRamScriptVersion	431
20.7.31	SDMA_PrepareTransfer	431
20.7.32	SDMA_PrepareP2PTransfer	432
20.7.33	SDMA_SubmitTransfer	433
20.7.34	SDMA_StartTransfer	433
20.7.35	SDMA_StopTransfer	433
20.7.36	SDMA_AbortTransfer	434
20.7.37	SDMA_GetTransferredBytes	434
20.7.38	SDMA_HandleIRQ	434

Chapter 21 SEMA4: Hardware Semaphores Driver

21.1	Overview	435
21.2	Macro Definition Documentation	436
21.2.1	SEMA4_GATE_NUM_RESET_ALL	436

Section No.	Title	Page No.
21.3	Function Documentation	436
21.3.1	SEMA4_Init	436
21.3.2	SEMA4_Deinit	436
21.3.3	SEMA4_TryLock	436
21.3.4	SEMA4_Lock	437
21.3.5	SEMA4_Unlock	437
21.3.6	SEMA4_GetLockProc	437
21.3.7	SEMA4_ResetGate	438
21.3.8	SEMA4_ResetAllGates	438
21.3.9	SEMA4_EnableGateNotifyInterrupt	439
21.3.10	SEMA4_DisableGateNotifyInterrupt	439
21.3.11	SEMA4_GetGateNotifyStatus	439
21.3.12	SEMA4_ResetGateNotify	440
21.3.13	SEMA4_ResetAllGateNotify	440
Chapter 22	TMU: Thermal Management Unit Driver	
22.1	Overview	442
22.2	Typical use case	442
22.2.1	Monitor and report Configuration	442
22.3	Data Structure Documentation	445
22.3.1	struct tmu_threshold_config_t	445
22.3.2	struct tmu_interrupt_status_t	446
22.3.3	struct tmu_config_t	446
22.4	Macro Definition Documentation	446
22.4.1	FSL_TMU_DRIVER_VERSION	446
22.5	Enumeration Type Documentation	446
22.5.1	_tmu_interrupt_enable	446
22.5.2	_tmu_interrupt_status_flags	447
22.5.3	tmu_average_low_pass_filter_t	447
22.5.4	tmu_amplifier_gain_t	447
22.5.5	tmu_amplifier_reference_voltage_t	448
22.6	Function Documentation	448
22.6.1	TMU_Init	448
22.6.2	TMU_Deinit	449
22.6.3	TMU_GetDefaultConfig	449
22.6.4	TMU_Enable	449
22.6.5	TMU_EnableInterrupts	449
22.6.6	TMU_DisableInterrupts	450
22.6.7	TMU_GetInterruptStatusFlags	450

Section No.	Title	Page No.
22.6.8	TMU_ClearInterruptStatusFlags	450
22.6.9	TMU_GetImmediateTemperature	450
22.6.10	TMU_GetAverageTemperature	451
22.6.11	TMU_SetHighTemperatureThresold	451
Chapter 23 WDOG: Watchdog Timer Driver		
23.1	Overview	453
23.2	Typical use case	453
23.3	Data Structure Documentation	454
23.3.1	struct wdog_work_mode_t	454
23.3.2	struct wdog_config_t	454
23.4	Enumeration Type Documentation	455
23.4.1	_wdog_interrupt_enable	455
23.4.2	_wdog_status_flags	455
23.5	Function Documentation	455
23.5.1	WDOG_GetDefaultConfig	455
23.5.2	WDOG_Init	456
23.5.3	WDOG_Deinit	456
23.5.4	WDOG_Enable	456
23.5.5	WDOG_Disable	457
23.5.6	WDOG_TriggerSystemSoftwareReset	457
23.5.7	WDOG_TriggerSoftwareSignal	457
23.5.8	WDOG_EnableInterrupts	458
23.5.9	WDOG_GetStatusFlags	459
23.5.10	WDOG_ClearInterruptStatus	459
23.5.11	WDOG_SetTimeoutValue	460
23.5.12	WDOG_SetInterrputTimeoutValue	460
23.5.13	WDOG_DisablePowerDownEnable	460
23.5.14	WDOG_Refresh	461
Chapter 24 Debug Console		
24.1	Overview	462
24.2	Function groups	462
24.2.1	Initialization	462
24.2.2	Advanced Feature	463
24.2.3	SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART	467
24.3	Typical use case	468

Section No.	Title	Page No.
24.4	Macro Definition Documentation	470
24.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	470
24.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	470
24.4.3	DEBUGCONSOLE_DISABLE	470
24.4.4	SDK_DEBUGCONSOLE	470
24.4.5	PRINTF	470
24.5	Function Documentation	470
24.5.1	DbgConsole_Init	470
24.5.2	DbgConsole_Deinit	471
24.5.3	DbgConsole_EnterLowpower	471
24.5.4	DbgConsole_ExitLowpower	472
24.5.5	DbgConsole_Printf	472
24.5.6	DbgConsole_Vprintf	472
24.5.7	DbgConsole_Putchar	472
24.5.8	DbgConsole_Scanf	473
24.5.9	DbgConsole_Getchar	473
24.5.10	DbgConsole_BlockingPrintf	474
24.5.11	DbgConsole_BlockingVprintf	474
24.5.12	DbgConsole_Flush	474
24.5.13	DbgConsole_TryGetchar	475
24.6	debug console configuration	477
24.6.1	Overview	477
24.6.2	Macro Definition Documentation	478
24.7	Semihosting	480
24.7.1	Guide Semihosting for IAR	480
24.7.2	Guide Semihosting for Keil μ Vision	480
24.7.3	Guide Semihosting for MCUXpresso IDE	481
24.7.4	Guide Semihosting for ARMGCC	481
24.8	SWO	484
24.8.1	Guide SWO for SDK	484
24.8.2	Guide SWO for Keil μ Vision	485
24.8.3	Guide SWO for MCUXpresso IDE	486
24.8.4	Guide SWO for ARMGCC	486
Chapter 25 CODEC Driver		
25.1	Overview	487
25.2	CODEC Common Driver	488
25.2.1	Overview	488
25.2.2	Data Structure Documentation	493

Section No.	Title	Page No.
25.2.3	Macro Definition Documentation	494
25.2.4	Enumeration Type Documentation	494
25.2.5	Function Documentation	499
25.3	CODEC I2C Driver	503
25.3.1	Overview	503
25.3.2	Data Structure Documentation	504
25.3.3	Enumeration Type Documentation	504
25.3.4	Function Documentation	504
25.4	AK4497 Driver	507
25.4.1	Overview	507
25.4.2	Data Structure Documentation	510
25.4.3	Macro Definition Documentation	510
25.4.4	Enumeration Type Documentation	510
25.4.5	Function Documentation	513
25.4.6	AK4497 Adapter	517
25.5	WM8524 Driver	525
25.5.1	Overview	525
25.5.2	Data Structure Documentation	526
25.5.3	Macro Definition Documentation	526
25.5.4	Typedef Documentation	526
25.5.5	Enumeration Type Documentation	526
25.5.6	Function Documentation	526
25.5.7	WM8524 Adapter	528
 Chapter 26 Serial Manager		
26.1	Overview	536
26.2	Data Structure Documentation	539
26.2.1	struct serial_manager_config_t	539
26.2.2	struct serial_manager_callback_message_t	540
26.3	Macro Definition Documentation	540
26.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE	540
26.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE	540
26.3.3	SERIAL_MANAGER_USE_COMMON_TASK	540
26.3.4	SERIAL_MANAGER_HANDLE_SIZE	540
26.3.5	SERIAL_MANAGER_HANDLE_DEFINE	540
26.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	541
26.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE	541
26.3.8	SERIAL_MANAGER_TASK_PRIORITY	542
26.3.9	SERIAL_MANAGER_TASK_STACK_SIZE	542

Section No.	Title	Page No.
26.4	Enumeration Type Documentation	542
26.4.1	serial_port_type_t	542
26.4.2	serial_manager_type_t	542
26.4.3	serial_manager_status_t	542
26.5	Function Documentation	543
26.5.1	SerialManager_Init	543
26.5.2	SerialManager_Deinit	544
26.5.3	SerialManager_OpenWriteHandle	544
26.5.4	SerialManager_CloseWriteHandle	546
26.5.5	SerialManager_OpenReadHandle	547
26.5.6	SerialManager_CloseReadHandle	548
26.5.7	SerialManager_WriteBlocking	548
26.5.8	SerialManager_ReadBlocking	549
26.5.9	SerialManager_WriteNonBlocking	550
26.5.10	SerialManager_ReadNonBlocking	550
26.5.11	SerialManager_TryRead	551
26.5.12	SerialManager_CancelWriting	552
26.5.13	SerialManager_CancelReading	552
26.5.14	SerialManager_InstallTxCallback	553
26.5.15	SerialManager_InstallRxCallback	553
26.5.16	SerialManager_needPollingIsr	555
26.5.17	SerialManager_EnterLowpower	555
26.5.18	SerialManager_ExitLowpower	555
26.5.19	SerialManager_SetLowpowerCriticalCb	556
26.6	Serial Port Uart	557
26.6.1	Overview	557
26.6.2	Enumeration Type Documentation	557
26.7	Serial Port SWO	558
26.7.1	Overview	558
26.7.2	Data Structure Documentation	558
26.7.3	Enumeration Type Documentation	558
Chapter 27 Enet_cmsis_driver		
27.1	Typical use case	559
27.1.1	CODEC Adapter	561

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.

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/.

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

MCUXpresso SDK Folder Structure

Chapter 2

Trademarks

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

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.

© 2021 NXP B.V.

Chapter 3

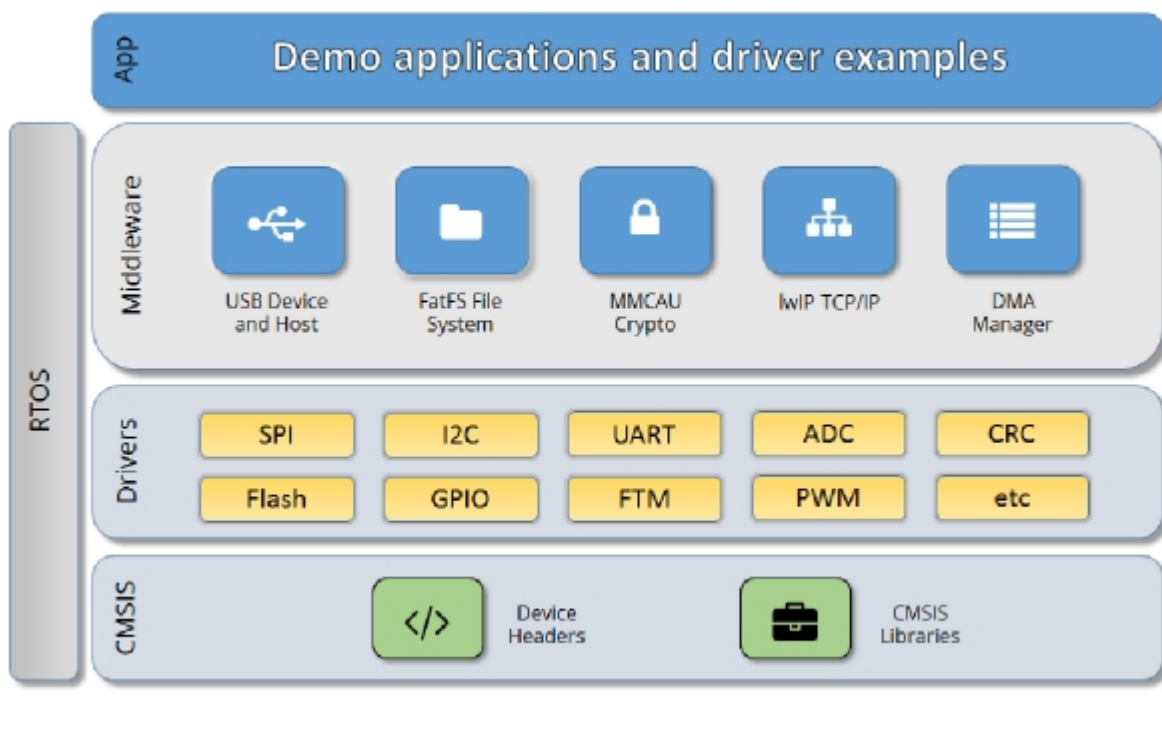
Architectural Overview

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

Overview

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

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



MCUXpresso SDK Block Diagram

MCU header files

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

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler
PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

The first level of the weak implementation are the functions defined in the vector table. In the devices/⟨DEVICE_NAME⟩/⟨TOOLCHAIN⟩/startup_⟨DEVICE_NAME⟩.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0_DriverIRQHandler) jumps to itself (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

Clock Driver

4.1 Overview

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

The clock driver supports:

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

Data Structures

- struct `ccm_analog_frac_pll_config_t`
Fractional-N PLL configuration. [More...](#)
- struct `ccm_analog_integer_pll_config_t`
Integer PLL configuration. [More...](#)

Macros

- #define `OSC24M_CLK_FREQ` 24000000U
XTAL 24M clock frequency.
- #define `CLKPAD_FREQ` 0U
pad clock frequency.
- #define `ECSPI_CLOCKS`
Clock ip name array for ECSPI.
- #define `ENET_CLOCKS`
Clock ip name array for ENET.
- #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 `IPMUX_CLOCKS`
Clock ip name array for IPMUX.
- #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**
Clock ip name array for USDHC.
- #define **WDOG_CLOCKS**
Clock ip name array for WDOG.
- #define **TMU_CLOCKS**
Clock ip name array for TEMPSSENSOR.
- #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 **PDM_CLOCKS**
Clock ip name array for PDM.
- #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 *)((uintptr_t)(root) + (off))))
CCM reg macros to map corresponding registers.
- #define **AUDIO_PLL1_GEN_CTRL_OFFSET** 0x00
CCM Analog registers offset.
- #define **CCM_ANALOG_TUPLE**(reg, shift) (((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 **CLOCK_ROOT_SOURCE**
clock root source
- #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`,
`kCLOCK_PerClk`,
`kCLOCK_EnetIpgClk`,
`kCLOCK_Osc24MClk`,
`kCLOCK_ArmPllClk`,
`kCLOCK_DramPllClk`,
`kCLOCK_SysPll1Clk`,
`kCLOCK_SysPll1Div2Clk`,
`kCLOCK_SysPll1Div3Clk`,
`kCLOCK_SysPll1Div4Clk`,
`kCLOCK_SysPll1Div5Clk`,
`kCLOCK_SysPll1Div6Clk`,
`kCLOCK_SysPll1Div8Clk`,
`kCLOCK_SysPll1Div10Clk`,
`kCLOCK_SysPll1Div20Clk`,
`kCLOCK_SysPll2Clk`,
`kCLOCK_SysPll2Div2Clk`,
`kCLOCK_SysPll2Div3Clk`,
`kCLOCK_SysPll2Div4Clk`,
`kCLOCK_SysPll2Div5Clk`,
`kCLOCK_SysPll2Div6Clk`,
`kCLOCK_SysPll2Div8Clk`,
`kCLOCK_SysPll2Div10Clk`,
`kCLOCK_SysPll2Div20Clk`,
`kCLOCK_SysPll3Clk`,
`kCLOCK_AudioPll1Clk`,
`kCLOCK_AudioPll2Clk`,
`kCLOCK_VideoPll1Clk`,
`kCLOCK_ExtClk1`,
`kCLOCK_ExtClk2`,
`kCLOCK_ExtClk3`,
`kCLOCK_ExtClk4`,
`kCLOCK_NoneName` }
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_Enet1 = CCM_TUPLE(10U, 17U),
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_Iomux = CCM_TUPLE(27U, 33U),
kCLOCK_Ipmux1 = CCM_TUPLE(28U, 33U),
kCLOCK_Ipmux2 = CCM_TUPLE(29U, 33U),
kCLOCK_Ipmux3 = CCM_TUPLE(30U, 33U),
kCLOCK_Ipmux4 = CCM_TUPLE(31U, 33U),
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, 16U),
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,
    kCLOCK_RootAxi = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[16].TARGET_ROOT),
    kCLOCK_RootEnetAxi = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[17].TARGET_ROOT),
    kCLOCK_RootNoc = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[26].TARGET_ROOT),
    kCLOCK_RootAhb = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[32].TARGET_ROOT),
    kCLOCK_RootIpg = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[33].TARGET_ROOT),
    kCLOCK_RootAudioAhb,
    kCLOCK_RootAudioIpg,
    kCLOCK_RootDramAlt,
    kCLOCK_RootSai1 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[75].TARGET_ROOT),
    kCLOCK_RootSai2 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[76].TARGET_ROOT),
    kCLOCK_RootSai3 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[77].TARGET_ROOT),
    kCLOCK_RootSai4 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[78].TARGET_ROOT),
    kCLOCK_RootSai5 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[79].TARGET_ROOT),
    kCLOCK_RootSai6 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[80].TARGET_ROOT),
    kCLOCK_RootEnetRef = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[83].TARGET_ROOT),
    kCLOCK_RootEnetTimer = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[84].TARGET_ROOT),
    kCLOCK_RootEnetPhy = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[85].TARGET_ROOT),
    kCLOCK_RootQspi = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[87].TARGET_ROOT),
    kCLOCK_RootI2c1 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[90].TARGET_ROOT),
    kCLOCK_RootI2c2 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[91].TARGET_ROOT),
    kCLOCK_RootI2c3 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[92].TARGET_ROOT),
    kCLOCK_RootI2c4 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[93].TARGET_ROOT)
}

```

```

T),
kCLOCK_RootUart1 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[94].TARGET_ROOT),
kCLOCK_RootUart2 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[95].TARGET_ROOT),
kCLOCK_RootUart3 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[96].TARGET_ROOT),
kCLOCK_RootUart4 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[97].TARGET_ROOT),
kCLOCK_RootEcspl,
kCLOCK_RootEcspl2,
kCLOCK_RootEcspl3,
kCLOCK_RootPwm1 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[103].TARGET_ROOT),
kCLOCK_RootPwm2 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[104].TARGET_ROOT),
kCLOCK_RootPwm3 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[105].TARGET_ROOT),
kCLOCK_RootPwm4 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[106].TARGET_ROOT),
kCLOCK_RootGpt1 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[107].TARGET_ROOT),
kCLOCK_RootGpt2 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[108].TARGET_ROOT),
kCLOCK_RootGpt3 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[109].TARGET_ROOT),
kCLOCK_RootGpt4 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[110].TARGET_ROOT),
kCLOCK_RootGpt5 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[111].TARGET_ROOT),
kCLOCK_RootGpt6 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[112].TARGET_ROOT),
kCLOCK_RootWdog = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[114].TARGET_ROOT),
kCLOCK_RootPdm = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[132].TARGET_ROOT) }

```

ccm root name used to get clock frequency.

- enum `clock_root_t` {

```

kCLOCK_M4ClkRoot = 0,
kCLOCK_AxiClkRoot,
kCLOCK_NocClkRoot,
kCLOCK_AhbClkRoot,
kCLOCK_IpgClkRoot,
kCLOCK_AudioAhbClkRoot,
kCLOCK_AudioIpgClkRoot,
kCLOCK_DramAltClkRoot,
kCLOCK_Sai1ClkRoot,
kCLOCK_Sai2ClkRoot,
kCLOCK_Sai3ClkRoot,
kCLOCK_Sai4ClkRoot,
kCLOCK_Sai5ClkRoot,
kCLOCK_Sai6ClkRoot,
kCLOCK_QspiClkRoot,
kCLOCK_I2c1ClkRoot,
kCLOCK_I2c2ClkRoot,
kCLOCK_I2c3ClkRoot,
kCLOCK_I2c4ClkRoot,
kCLOCK_Uart1ClkRoot,
kCLOCK_Uart2ClkRoot,
kCLOCK_Uart3ClkRoot,
kCLOCK_Uart4ClkRoot,
kCLOCK_Ecspi1ClkRoot,
kCLOCK_Ecspi2ClkRoot,
kCLOCK_Ecspi3ClkRoot,
kCLOCK_Pwm1ClkRoot,
kCLOCK_Pwm2ClkRoot,
kCLOCK_Pwm3ClkRoot,
kCLOCK_Pwm4ClkRoot,
kCLOCK_Gpt1ClkRoot,
kCLOCK_Gpt2ClkRoot,
kCLOCK_Gpt3ClkRoot,
kCLOCK_Gpt4ClkRoot,
kCLOCK_Gpt5ClkRoot,
kCLOCK_Gpt6ClkRoot,
kCLOCK_WdogClkRoot,
kCLOCK_PdmClkRoot }

```

ccm clock root used to get clock frequency.

- enum `clock_rootmux_m4_clk_sel_t` {

```

kCLOCK_M4RootmuxOsc24M = 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_AxiRootmuxOsc24M` = 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_AhbRootmuxOsc24M` = 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_audio_ahb_clk_sel_t` {
`kCLOCK_AudioAhbRootmuxOsc24M` = 0U,
`kCLOCK_AudioAhbRootmuxSysPll2Div2` = 1U,
`kCLOCK_AudioAhbRootmuxSysPll1` = 2U,
`kCLOCK_AudioAhbRootmuxSysPll2` = 3U,
`kCLOCK_AudioAhbRootmuxSysPll2Div6` = 4U,
`kCLOCK_AudioAhbRootmuxSysPll3` = 5U,
`kCLOCK_AudioAhbRootmuxAudioPll1` = 6U,
`kCLOCK_AudioAhbRootmuxVideoPll1` = 7U }

Root clock select enumeration for Audio AHB bus.

- enum `clock_rootmux_qspi_clk_sel_t` {

```

kCLOCK_QspiRootmuxOsc24M = 0U,
kCLOCK_QspiRootmuxSysPll1Div2 = 1U,
kCLOCK_QspiRootmuxSysPll2Div3 = 2U,
kCLOCK_QspiRootmuxSysPll2Div2 = 3U,
kCLOCK_QspiRootmuxAudioPll2 = 4U,
kCLOCK_QspiRootmuxSysPll1Div3 = 5U,
kCLOCK_QspiRootmuxSysPll3 = 6,
kCLOCK_QspiRootmuxSysPll1Div8 = 7U }

```

Root clock select enumeration for QSPI peripheral.

- enum `clock_rootmux_ecspi_clk_sel_t` {
`kCLOCK_EcspiRootmuxOsc24M = 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_enet_axi_clk_sel_t` {
`kCLOCK_EnetAxiRootmuxOsc24M = 0U,`
`kCLOCK_EnetAxiRootmuxSysPll1Div3 = 1U,`
`kCLOCK_EnetAxiRootmuxSysPll1 = 2U,`
`kCLOCK_EnetAxiRootmuxSysPll2Div4 = 3U,`
`kCLOCK_EnetAxiRootmuxSysPll2Div5 = 4U,`
`kCLOCK_EnetAxiRootmuxAudioPll1 = 5U,`
`kCLOCK_EnetAxiRootmuxVideoPll1 = 6U,`
`kCLOCK_EnetAxiRootmuxSysPll3 = 7U }`

Root clock select enumeration for ENET AXI bus.

- enum `clock_rootmux_enet_ref_clk_sel_t` {
`kCLOCK_EnetRefRootmuxOsc24M = 0U,`
`kCLOCK_EnetRefRootmuxSysPll2Div8 = 1U,`
`kCLOCK_EnetRefRootmuxSysPll2Div20 = 2U,`
`kCLOCK_EnetRefRootmuxSysPll2Div10 = 3U,`
`kCLOCK_EnetRefRootmuxSysPll1Div5 = 4U,`
`kCLOCK_EnetRefRootmuxAudioPll1 = 5U,`
`kCLOCK_EnetRefRootmuxVideoPll1 = 6U,`
`kCLOCK_EnetRefRootmuxExtClk4 = 7U }`

Root clock select enumeration for ENET REF Clcok.

- enum `clock_rootmux_enet_timer_clk_sel_t` {

```

kCLOCK_EnetTimerRootmuxOsc24M = 0U,
kCLOCK_EnetTimerRootmuxSysPll2Div10 = 1U,
kCLOCK_EnetTimerRootmuxAudioPll1 = 2U,
kCLOCK_EnetTimerRootmuxExtClk1 = 3U,
kCLOCK_EnetTimerRootmuxExtClk2 = 4U,
kCLOCK_EnetTimerRootmuxExtClk3 = 5U,
kCLOCK_EnetTimerRootmuxExtClk4 = 6U,
kCLOCK_EnetTimerRootmuxVideoPll1 = 7U }

```

Root clock select enumeration for ENET TIMER Clcok.

- enum `clock_rootmux_enet_phy_clk_sel_t` {
`kCLOCK_EnetPhyRootmuxOsc24M` = 0U,
`kCLOCK_EnetPhyRootmuxSysPll2Div20` = 1U,
`kCLOCK_EnetPhyRootmuxSysPll2Div8` = 2U,
`kCLOCK_EnetPhyRootmuxSysPll2Div5` = 3U,
`kCLOCK_EnetPhyRootmuxSysPll2Div2` = 4U,
`kCLOCK_EnetPhyRootmuxAudioPll1` = 5U,
`kCLOCK_EnetPhyRootmuxVideoPll1` = 6U,
`kCLOCK_EnetPhyRootmuxAudioPll2` = 7U }

Root clock select enumeration for ENET PHY Clcok.

- enum `clock_rootmux_i2c_clk_sel_t` {
`kCLOCK_I2cRootmuxOsc24M` = 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_UartRootmuxOsc24M` = 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_GptRootmuxOsc24M = 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_WdogRootmuxOsc24M = 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_PwmRootmuxOsc24M = 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_SaiRootmuxOsc24M = 0U,`
`kCLOCK_SaiRootmuxAudioPll1 = 1U,`
`kCLOCK_SaiRootmuxAudioPll2 = 2U,`
`kCLOCK_SaiRootmuxVideoPll1 = 3U,`
`kCLOCK_SaiRootmuxSysPll1Div6 = 4U,`
`kCLOCK_SaiRootmuxOsc26m = 5U,`
`kCLOCK_SaiRootmuxExtClk1 = 6U,`
`kCLOCK_SaiRootmuxExtClk2 = 7U }`

Root clock select enumeration for SAI peripheral.

- enum `clock_rootmux_pdm_clk_sel_t` {


```

kCLOCK_PdmRootmuxOsc24M = 0U,
kCLOCK_PdmRootmuxSystemPll2 = 1U,
kCLOCK_PdmRootmuxAudioPll1 = 2U,
kCLOCK_PdmRootmuxSysPll1 = 3U,
kCLOCK_PdmRootmuxSysPll2 = 4U,
kCLOCK_PdmRootmuxSysPll3 = 5U,
kCLOCK_PdmRootmuxExtClk3 = 6U,
kCLOCK_PdmRootmuxAudioPll2 = 7U }

```

Root clock select enumeration for PDM peripheral.

- enum `clock_rootmux_noc_clk_sel_t` {
`kCLOCK_NocRootmuxOsc24M = 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 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[12].PLL_CTRL),`
`kCLOCK_GpuPllGate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[13].PLL_CTRL),`
`kCLOCK_VpuPllGate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[14].PLL_CTRL),`
`kCLOCK_DramPllGate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[15].PLL_CTRL),`
`kCLOCK_SysPll1Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[16].PLL_CTRL),`
`kCLOCK_SysPll1Div2Gate,`
`kCLOCK_SysPll1Div3Gate,`
`kCLOCK_SysPll1Div4Gate,`
`kCLOCK_SysPll1Div5Gate,`
`kCLOCK_SysPll1Div6Gate,`
`kCLOCK_SysPll1Div8Gate,`
`kCLOCK_SysPll1Div10Gate,`
`kCLOCK_SysPll1Div20Gate,`
`kCLOCK_SysPll2Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[25].PLL_CTRL)`

- ```

 TRL),
 kCLOCK_SysPll2Div2Gate,
 kCLOCK_SysPll2Div3Gate,
 kCLOCK_SysPll2Div4Gate,
 kCLOCK_SysPll2Div5Gate,
 kCLOCK_SysPll2Div6Gate,
 kCLOCK_SysPll2Div8Gate,
 kCLOCK_SysPll2Div10Gate,
 kCLOCK_SysPll2Div20Gate,
 kCLOCK_SysPll3Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[34].PLL_CTRL),
 kCLOCK_AudioPll1Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[35].PLL_CTRL),
 kCLOCK_AudioPll2Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[36].PLL_CTRL),
 kCLOCK_VideoPll1Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[37].PLL_CTRL),
 kCLOCK_VideoPll2Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, 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_DramPllInternalPll1BypassCtrl,
 kCLOCK_GpuPLLpwrBypassCtrl,
 kCLOCK_VpuPllPwrBypassCtrl,
 kCLOCK_ArmPllPwrBypassCtrl,
 kCLOCK_SysPll1InternalPll1BypassCtrl,
 kCLOCK_SysPll2InternalPll1BypassCtrl,
 kCLOCK_SysPll3InternalPll1BypassCtrl }

```
- PLL control names for PLL bypass.*
- enum `clock_pll_clke_t` {

```

kCLOCK_AudioPll1Clke,
kCLOCK_AudioPll2Clke,
kCLOCK_VideoPll1Clke,
kCLOCK_DramPllClke,
kCLOCK_GpuPllClke,
kCLOCK_VpuPllClke,
kCLOCK_ArmPllClke,
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 }

```

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

- enum `clock_pll_ctrl_t`  
*ANALOG Power down override control.*
- enum {  
    `kANALOG_PllRefOsc24M` = 0U,  
    `kANALOG_PllPadClk` = 1U }  
*PLL reference clock select.*

## Driver version

- #define `FSL_CLOCK_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 0)`)  
*CLOCK driver version 2.4.0.*

## 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.*

## CCM Gate Control

- static void **CLOCK\_ControlGate** (uintptr\_t ccmGate, clock\_gate\_value\_t control)  
*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\_OverridePllClke** (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\_integer\_pll\_config\_t** \*config)
- *Initializes the ANALOG ARM PLL.*
- void **CLOCK\_DeinitArmPll** (void)
- *De-initialize the ARM PLL.*
- void **CLOCK\_InitSysPll1** (const **ccm\_analog\_integer\_pll\_config\_t** \*config)
- *Initializes the ANALOG SYS PLL1.*
- void **CLOCK\_DeinitSysPll1** (void)
- *De-initialize the System PLL1.*
- void **CLOCK\_InitSysPll2** (const **ccm\_analog\_integer\_pll\_config\_t** \*config)
- *Initializes the ANALOG SYS PLL2.*
- void **CLOCK\_DeinitSysPll2** (void)
- *De-initialize the System PLL2.*
- void **CLOCK\_InitSysPll3** (const **ccm\_analog\_integer\_pll\_config\_t** \*config)
- *Initializes the ANALOG SYS PLL3.*
- void **CLOCK\_DeinitSysPll3** (void)
- *De-initialize the System PLL3.*
- 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\_InitIntegerPll** (CCM\_ANALOG\_Type \*base, const **ccm\_analog\_integer\_pll\_config\_t** \*config, **clock\_pll\_ctrl\_t** type)
- *Initializes the ANALOG Integer PLL.*
- uint32\_t **CLOCK\_GetIntegerPllFreq** (CCM\_ANALOG\_Type \*base, **clock\_pll\_ctrl\_t** type, uint32\_t refClkFreq, bool pll1Bypass)
- *Get the ANALOG Integer 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\\_GetClockRootFreq](#) ([clock\\_root\\_t](#) clockRoot)  
*Gets the frequency of selected clock root.*
- 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.*
- uint32\_t [CLOCK\\_GetEnetAxiFreq](#) (void)  
*brief Get the CCM Enet AXI bus frequency.*

## 4.2 Data Structure Documentation

### 4.2.1 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*
- uint32\_t [mainDiv](#)  
*Value of the 10-bit programmable main-divider, range must be 64~1023.*
- uint32\_t [dsm](#)  
*Value of 16-bit DSM.*
- uint8\_t [preDiv](#)  
*Value of the 6-bit programmable pre-divider, range must be 1~63.*
- uint8\_t [postDiv](#)  
*Value of the 3-bit programmable Scaler, range must be 0~6.*

### 4.2.2 struct ccm\_analog\_integer\_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*
- uint32\_t [mainDiv](#)  
*Value of the 10-bit programmable main-divider, range must be 64~1023.*

- uint8\_t `preDiv`  
Value of the 6-bit programmable pre-divider, range must be 1~63.
- uint8\_t `postDiv`  
Value of the 3-bit programmable Scaler, range must be 0~6.

## 4.3 Macro Definition Documentation

### 4.3.1 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))

### 4.3.2 #define ECSPI\_CLOCKS

Value:

```
{
 kCLOCK_IpInvalid, kCLOCK_Ecspl1, kCLOCK_Ecspl2, \
 kCLOCK_Ecspl3, \
}
```

### 4.3.3 #define ENET\_CLOCKS

Value:

```
{
 \
 kCLOCK_Enet1, \
}
```

### 4.3.4 #define GPIO\_CLOCKS

Value:

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

### 4.3.5 #define GPT\_CLOCKS

Value:

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

### 4.3.6 #define I2C\_CLOCKS

Value:

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

### 4.3.7 #define IOMUX\_CLOCKS

Value:

```
{
 \
 kCLOCK_Iomux, \
}
```

### 4.3.8 #define IPMUX\_CLOCKS

Value:

```
{
 \
 kCLOCK_Ipmux1, kCLOCK_Ipmux2,
 kCLOCK_Ipmux3, kCLOCK_Ipmux4, \
}
```

### 4.3.9 #define PWM\_CLOCKS

Value:

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

### 4.3.10 #define RDC\_CLOCKS

Value:

```
{
 \
 kCLOCK_Rdc, \
}
```



### 4.3.11 #define SAI\_CLOCKS

Value:

```

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

```

### 4.3.12 #define RDC\_SEMA42\_CLOCKS

Value:

```

{
 kCLOCK_IpInvalid, kCLOCK_Sema42_1, kCLOCK_Sema42_2 \
}

```

### 4.3.13 #define UART\_CLOCKS

Value:

```

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

```

### 4.3.14 #define USDHC\_CLOCKS

Value:

```

{
 kCLOCK_IpInvalid, kCLOCK_Usdhc1, kCLOCK_Usdhc2, \
 kCLOCK_Usdhc3 \
}

```

### 4.3.15 #define WDOG\_CLOCKS

Value:

```

{
 kCLOCK_IpInvalid, kCLOCK_Wdog1, kCLOCK_Wdog2, \
 kCLOCK_Wdog3 \
}

```

#### 4.3.16 #define TMU\_CLOCKS

Value:

```
{
 kCLOCK_TempSensor, \
}
```

#### 4.3.17 #define SDMA\_CLOCKS

Value:

```
{
 kCLOCK_Sdma1, kCLOCK_Sdma2, kCLOCK_Sdma3 \
}
```

#### 4.3.18 #define MU\_CLOCKS

Value:

```
{
 kCLOCK_Mu \
}
```

#### 4.3.19 #define QSPI\_CLOCKS

Value:

```
{
 kCLOCK_Qspi \
}
```

#### 4.3.20 #define PDM\_CLOCKS

Value:

```
{
 kCLOCK_Pdm \
}
```

**4.3.21 #define kCLOCK\_CoreSysClk kCLOCK\_CoreM4Clk**

**4.3.22 #define CLOCK\_GetCoreSysClkFreq CLOCK\_GetCoreM4Freq**

## 4.4 Enumeration Type Documentation

### 4.4.1 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.  
*kCLOCK\_PerClk* Peripheral Clock.  
*kCLOCK\_EnetIpgClk* ENET IPG Clock.  
*kCLOCK\_Osc24MClk* OSC 24M clock.  
*kCLOCK\_ArmPllClk* Arm PLL clock.  
*kCLOCK\_DramPllClk* Dram PLL clock.  
*kCLOCK\_SysPll1Clk* Sys PLL1 clock.  
*kCLOCK\_SysPll1Div2Clk* Sys PLL1 clock divided by 2.  
*kCLOCK\_SysPll1Div3Clk* Sys PLL1 clock divided by 3.  
*kCLOCK\_SysPll1Div4Clk* Sys PLL1 clock divided by 4.  
*kCLOCK\_SysPll1Div5Clk* Sys PLL1 clock divided by 5.  
*kCLOCK\_SysPll1Div6Clk* Sys PLL1 clock divided by 6.  
*kCLOCK\_SysPll1Div8Clk* Sys PLL1 clock divided by 8.  
*kCLOCK\_SysPll1Div10Clk* Sys PLL1 clock divided by 10.  
*kCLOCK\_SysPll1Div20Clk* Sys PLL1 clock divided by 20.  
*kCLOCK\_SysPll2Clk* Sys PLL2 clock.  
*kCLOCK\_SysPll2Div2Clk* Sys PLL2 clock divided by 2.  
*kCLOCK\_SysPll2Div3Clk* Sys PLL2 clock divided by 3.  
*kCLOCK\_SysPll2Div4Clk* Sys PLL2 clock divided by 4.  
*kCLOCK\_SysPll2Div5Clk* Sys PLL2 clock divided by 5.  
*kCLOCK\_SysPll2Div6Clk* Sys PLL2 clock divided by 6.  
*kCLOCK\_SysPll2Div8Clk* Sys PLL2 clock divided by 8.  
*kCLOCK\_SysPll2Div10Clk* Sys PLL2 clock divided by 10.  
*kCLOCK\_SysPll2Div20Clk* Sys PLL2 clock divided by 20.  
*kCLOCK\_SysPll3Clk* Sys PLL3 clock.  
*kCLOCK\_AudioPll1Clk* Audio PLL1 clock.  
*kCLOCK\_AudioPll2Clk* Audio PLL2 clock.  
*kCLOCK\_VideoPll1Clk* Video PLL1 clock.  
*kCLOCK\_ExtClk1* External clock1.  
*kCLOCK\_ExtClk2* External clock2.  
*kCLOCK\_ExtClk3* External clock3.  
*kCLOCK\_ExtClk4* External clock4.

***kCLOCK\_NoneName*** None Clock Name.

#### 4.4.2 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\_Enet1*** ENET1 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\_Iomux*** IOMUX Clock Gate.  
***kCLOCK\_Ipmux1*** IPMUX1 Clock Gate.  
***kCLOCK\_Ipmux2*** IPMUX2 Clock Gate.  
***kCLOCK\_Ipmux3*** IPMUX3 Clock Gate.  
***kCLOCK\_Ipmux4*** IPMUX4 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.  
***kCLOCK\_Uart2*** UART2 Clock Gate.  
***kCLOCK\_Uart3*** UART3 Clock Gate.  
***kCLOCK\_Uart4*** UART4 Clock Gate.  
***kCLOCK\_Usdhc1*** USDHC1 Clock Gate.  
***kCLOCK\_Usdhc2*** USDHC2 Clock Gate.  
***kCLOCK\_Wdog1*** WDOG1 Clock Gate.  
***kCLOCK\_Wdog2*** WDOG2 Clock Gate.  
***kCLOCK\_Wdog3*** WDOG3 Clock Gate.  
***kCLOCK\_Pdm*** PDM Clock Gate.  
***kCLOCK\_Usdhc3*** USDHC3 Clock Gate.  
***kCLOCK\_Sdma3*** SDMA3 Clock Gate.  
***kCLOCK\_TempSensor*** TempSensor Clock Gate.

#### 4.4.3 enum clock\_root\_control\_t

Enumerator

***kCLOCK\_RootM4*** ARM Cortex-M4 Clock control name.  
***kCLOCK\_RootAxi*** AXI Clock control name.  
***kCLOCK\_RootEnetAxi*** ENET AXI Clock control name.  
***kCLOCK\_RootNoc*** NOC Clock control name.  
***kCLOCK\_RootAhb*** AHB Clock control name.  
***kCLOCK\_RootIpg*** IPG Clock control name.  
***kCLOCK\_RootAudioAhb*** Audio AHB Clock control name.  
***kCLOCK\_RootAudioIpg*** Audio 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\_RootEnetRef*** ENET Clock control name.  
***kCLOCK\_RootEnetTimer*** ENET TIMER Clock control name.  
***kCLOCK\_RootEnetPhy*** ENET PHY 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.  
***kCLOCK\_RootPdm*** PDM Clock control name.

#### 4.4.4 enum clock\_root\_t

Enumerator

***kCLOCK\_M4ClkRoot*** ARM Cortex-M4 Clock control name.  
***kCLOCK\_AxiClkRoot*** AXI Clock control name.  
***kCLOCK\_NocClkRoot*** NOC Clock control name.  
***kCLOCK\_AhbClkRoot*** AHB Clock control name.  
***kCLOCK\_IpgClkRoot*** IPG Clock control name.  
***kCLOCK\_AudioAhbClkRoot*** Audio AHB Clock control name.  
***kCLOCK\_AudioIpgClkRoot*** Audio IPG Clock control name.  
***kCLOCK\_DramAltClkRoot*** DRAM ALT Clock control name.  
***kCLOCK\_Sai1ClkRoot*** SAI1 Clock control name.  
***kCLOCK\_Sai2ClkRoot*** SAI2 Clock control name.

***kCLOCK\_Sai3ClkRoot*** SAI3 Clock control name.  
***kCLOCK\_Sai4ClkRoot*** SAI4 Clock control name.  
***kCLOCK\_Sai5ClkRoot*** SAI5 Clock control name.  
***kCLOCK\_Sai6ClkRoot*** SAI6 Clock control name.  
***kCLOCK\_QspiClkRoot*** QSPI Clock control name.  
***kCLOCK\_I2c1ClkRoot*** I2C1 Clock control name.  
***kCLOCK\_I2c2ClkRoot*** I2C2 Clock control name.  
***kCLOCK\_I2c3ClkRoot*** I2C3 Clock control name.  
***kCLOCK\_I2c4ClkRoot*** I2C4 Clock control name.  
***kCLOCK\_Uart1ClkRoot*** UART1 Clock control name.  
***kCLOCK\_Uart2ClkRoot*** UART2 Clock control name.  
***kCLOCK\_Uart3ClkRoot*** UART3 Clock control name.  
***kCLOCK\_Uart4ClkRoot*** UART4 Clock control name.  
***kCLOCK\_Ecspi1ClkRoot*** ECSPi1 Clock control name.  
***kCLOCK\_Ecspi2ClkRoot*** ECSPi2 Clock control name.  
***kCLOCK\_Ecspi3ClkRoot*** ECSPi3 Clock control name.  
***kCLOCK\_Pwm1ClkRoot*** PWM1 Clock control name.  
***kCLOCK\_Pwm2ClkRoot*** PWM2 Clock control name.  
***kCLOCK\_Pwm3ClkRoot*** PWM3 Clock control name.  
***kCLOCK\_Pwm4ClkRoot*** PWM4 Clock control name.  
***kCLOCK\_Gpt1ClkRoot*** GPT1 Clock control name.  
***kCLOCK\_Gpt2ClkRoot*** GPT2 Clock control name.  
***kCLOCK\_Gpt3ClkRoot*** GPT3 Clock control name.  
***kCLOCK\_Gpt4ClkRoot*** GPT4 Clock control name.  
***kCLOCK\_Gpt5ClkRoot*** GPT5 Clock control name.  
***kCLOCK\_Gpt6ClkRoot*** GPT6 Clock control name.  
***kCLOCK\_WdogClkRoot*** WDOG Clock control name.  
***kCLOCK\_PdmClkRoot*** PDM Clock control name.

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

Enumerator

***kCLOCK\_M4RootmuxOsc24M*** ARM Cortex-M4 Clock from OSC 24M.  
***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.

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

Enumerator

*kCLOCK\_AxiRootmuxOsc24M* ARM AXI Clock from OSC 24M.  
*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.

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

Enumerator

*kCLOCK\_AhbRootmuxOsc24M* ARM AHB Clock from OSC 24M.  
*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.

#### 4.4.8 enum clock\_rootmux\_audio\_ahb\_clk\_sel\_t

Enumerator

*kCLOCK\_AudioAhbRootmuxOsc24M* ARM Audio AHB Clock from OSC 24M.  
*kCLOCK\_AudioAhbRootmuxSysPll2Div2* ARM Audio AHB Clock from SYSTEM PLL2 divided by 2.  
*kCLOCK\_AudioAhbRootmuxSysPll1* ARM Audio AHB Clock from SYSTEM PLL1.  
*kCLOCK\_AudioAhbRootmuxSysPll2* ARM Audio AHB Clock from SYSTEM PLL2.  
*kCLOCK\_AudioAhbRootmuxSysPll2Div6* ARM Audio AHB Clock from SYSTEM PLL2 divided by 6.  
*kCLOCK\_AudioAhbRootmuxSysPll3* ARM Audio AHB Clock from SYSTEM PLL3.  
*kCLOCK\_AudioAhbRootmuxAudioPll1* ARM Audio AHB Clock from AUDIO PLL1.  
*kCLOCK\_AudioAhbRootmuxVideoPll1* ARM Audio AHB Clock from VIDEO PLL1.



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

Enumerator

*kCLOCK\_QspiRootmuxOsc24M* ARM QSPI Clock from OSC 24M.  
*kCLOCK\_QspiRootmuxSysPll1Div2* ARM QSPI Clock from SYSTEM PLL1 divided by 2.  
*kCLOCK\_QspiRootmuxSysPll2Div3* ARM QSPI Clock from SYSTEM PLL2 divided by 3.  
*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.

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

Enumerator

*kCLOCK\_EcspiRootmuxOsc24M* ECSPI Clock from OSC 24M.  
*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.

#### 4.4.11 enum clock\_rootmux\_enet\_axi\_clk\_sel\_t

Enumerator

*kCLOCK\_EnetAxiRootmuxOsc24M* ENET AXI Clock from OSC 24M.  
*kCLOCK\_EnetAxiRootmuxSysPll1Div3* ENET AXI Clock from SYSTEM PLL1 divided by 3.  
*kCLOCK\_EnetAxiRootmuxSysPll1* ENET AXI Clock from SYSTEM PLL1.  
*kCLOCK\_EnetAxiRootmuxSysPll2Div4* ENET AXI Clock from SYSTEM PLL2 divided by 4.  
*kCLOCK\_EnetAxiRootmuxSysPll2Div5* ENET AXI Clock from SYSTEM PLL2 divided by 5.  
*kCLOCK\_EnetAxiRootmuxAudioPll1* ENET AXI Clock from AUDIO PLL1.  
*kCLOCK\_EnetAxiRootmuxVideoPll1* ENET AXI Clock from VIDEO PLL1.  
*kCLOCK\_EnetAxiRootmuxSysPll3* ENET AXI Clock from SYSTEM PLL3.

#### 4.4.12 enum clock\_rootmux\_enet\_ref\_clk\_sel\_t

Enumerator

***kCLOCK\_EnetRefRootmuxOsc24M*** ENET REF Clock from OSC 24M.  
***kCLOCK\_EnetRefRootmuxSysPll2Div8*** ENET REF Clock from SYSTEM PLL2 divided by 8.  
***kCLOCK\_EnetRefRootmuxSysPll2Div20*** ENET REF Clock from SYSTEM PLL2 divided by 20.  
***kCLOCK\_EnetRefRootmuxSysPll2Div10*** ENET REF Clock from SYSTEM PLL2 divided by 10.  
***kCLOCK\_EnetRefRootmuxSysPll1Div5*** ENET REF Clock from SYSTEM PLL1 divided by 5.  
***kCLOCK\_EnetRefRootmuxAudioPll1*** ENET REF Clock from AUDIO PLL1.  
***kCLOCK\_EnetRefRootmuxVideoPll1*** ENET REF Clock from VIDEO PLL1.  
***kCLOCK\_EnetRefRootmuxExtClk4*** ENET REF Clock from External Clock 4.

#### 4.4.13 enum clock\_rootmux\_enet\_timer\_clk\_sel\_t

Enumerator

***kCLOCK\_EnetTimerRootmuxOsc24M*** ENET TIMER Clock from OSC 24M.  
***kCLOCK\_EnetTimerRootmuxSysPll2Div10*** ENET TIMER Clock from SYSTEM PLL2 divided by 10.  
***kCLOCK\_EnetTimerRootmuxAudioPll1*** ENET TIMER Clock from AUDIO PLL1.  
***kCLOCK\_EnetTimerRootmuxExtClk1*** ENET TIMER Clock from External Clock 1.  
***kCLOCK\_EnetTimerRootmuxExtClk2*** ENET TIMER Clock External Clock 2.  
***kCLOCK\_EnetTimerRootmuxExtClk3*** ENET TIMER Clock from External Clock 3.  
***kCLOCK\_EnetTimerRootmuxExtClk4*** ENET TIMER Clock from External Clock 4.  
***kCLOCK\_EnetTimerRootmuxVideoPll1*** ENET TIMER Clock from VIDEO PLL1.

#### 4.4.14 enum clock\_rootmux\_enet\_phy\_clk\_sel\_t

Enumerator

***kCLOCK\_EnetPhyRootmuxOsc24M*** ENET PHY Clock from OSC 24M.  
***kCLOCK\_EnetPhyRootmuxSysPll2Div20*** ENET PHY Clock from SYSTEM PLL2 divided by 20.  
***kCLOCK\_EnetPhyRootmuxSysPll2Div8*** ENET PHY Clock from SYSTEM PLL2 divided by 8.  
***kCLOCK\_EnetPhyRootmuxSysPll2Div5*** ENET PHY Clock from SYSTEM PLL2 divided by 5.  
***kCLOCK\_EnetPhyRootmuxSysPll2Div2*** ENET PHY Clock from SYSTEM PLL2 divided by 2.  
***kCLOCK\_EnetPhyRootmuxAudioPll1*** ENET PHY Clock from AUDIO PLL1.  
***kCLOCK\_EnetPhyRootmuxVideoPll1*** ENET PHY Clock from VIDEO PLL1.  
***kCLOCK\_EnetPhyRootmuxAudioPll2*** ENET PHY Clock from AUDIO PLL2.

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

Enumerator

*kCLOCK\_I2cRootmuxOsc24M* I2C Clock from OSC 24M.  
*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.

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

Enumerator

*kCLOCK\_UartRootmuxOsc24M* UART Clock from OSC 24M.  
*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.

#### 4.4.17 enum clock\_rootmux\_gpt\_t

Enumerator

*kCLOCK\_GptRootmuxOsc24M* GPT Clock from OSC 24M.  
*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.

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

Enumerator

*kCLOCK\_WdogRootmuxOsc24M* WDOG Clock from OSC 24M.  
*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.

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

Enumerator

*kCLOCK\_PwmRootmuxOsc24M* PWM Clock from OSC 24M.  
*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.

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

Enumerator

*kCLOCK\_SaiRootmuxOsc24M* SAI Clock from OSC 24M.  
*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\_SaiRootmuxOsc26m* SAI Clock from OSC HDMI 26M.  
*kCLOCK\_SaiRootmuxExtClk1* SAI Clock from External Clock1, External Clock2, External Clock3.  
*kCLOCK\_SaiRootmuxExtClk2* SAI Clock from External Clock2, External Clock3, External Clock4.

#### 4.4.21 enum clock\_rootmux\_pdm\_clk\_sel\_t

Enumerator

***kCLOCK\_PdmRootmuxOsc24M*** GPT Clock from OSC 24M.  
***kCLOCK\_PdmRootmuxSystemPll2*** GPT Clock from SYSTEM PLL2 divided by 10.  
***kCLOCK\_PdmRootmuxAudioPll1*** GPT Clock from SYSTEM PLL1 divided by 2.  
***kCLOCK\_PdmRootmuxSysPll1*** GPT Clock from SYSTEM PLL1 divided by 20.  
***kCLOCK\_PdmRootmuxSysPll2*** GPT Clock from VIDEO PLL1.  
***kCLOCK\_PdmRootmuxSysPll3*** GPT Clock from SYSTEM PLL1 divided by 10.  
***kCLOCK\_PdmRootmuxExtClk3*** GPT Clock from AUDIO PLL1.  
***kCLOCK\_PdmRootmuxAudioPll2*** GPT Clock from External Clock1, External Clock2, External Clock3.

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

Enumerator

***kCLOCK\_NocRootmuxOsc24M*** NOC Clock from OSC 24M.  
***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.

#### 4.4.23 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.

#### 4.4.24 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.

#### 4.4.25 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\_DramPllInternalPll1BypassCtrl*** CCM DRAM PLL 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 bypass Control.  
***kCLOCK\_SysPll2InternalPll1BypassCtrl*** CCM System PLL2 bypass Control.  
***kCLOCK\_SysPll3InternalPll1BypassCtrl*** CCM System PLL3 bypass Control.

#### 4.4.26 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\_DramPllClke* Dram pll 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.

#### 4.4.27 anonymous enum

Enumerator

*kANALOG\_PllRefOsc24M* reference OSC 24M  
*kANALOG\_PllPadClk* reference PAD CLK

## 4.5 Function Documentation

**4.5.1 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).         |

#### 4.5.2 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).

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

## Parameters

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

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

## Parameters

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

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

## Parameters

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

#### 4.5.6 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>mux</i>        | Root mux value (see <a href="#">_ccm_rootmux_xxx</a> enumeration)   |
| <i>pre</i>        | Pre divider value (0-7, divider=n+1)                                |
| <i>post</i>       | Post divider value (0-63, divider=n+1)                              |

#### 4.5.7 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)                                           |

#### 4.5.8 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.

#### 4.5.9 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.

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

## 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> )                                        |

#### 4.5.11 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). |
|----------------|---------------------------------------------------------------------------------|

#### 4.5.12 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). |
|----------------|---------------------------------------------------------------------------------|

#### 4.5.13 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) |

#### 4.5.14 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) |

#### 4.5.15 static void CLOCK\_SetPllBypass ( 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 <code>ccm_analog_pll_control_t</code> 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> |

#### 4.5.16 **static bool CLOCK\_IsPllBypassed ( 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 <code>ccm_analog_pll_control_t</code> enumeration) |

Returns

PLL bypass status.

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

#### 4.5.17 **static bool CLOCK\_IsPllLocked ( 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) |

Returns

PLL lock status.

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

#### 4.5.18 **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 <code>ccm_analog_pll_clock_t</code> enumeration) |

#### 4.5.19 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 <code>ccm_analog_pll_clock_t</code> enumeration) |

#### 4.5.20 static void CLOCK\_OverridePllClke ( 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> |

#### 4.5.21 static void CLOCK\_OverridePllPd ( 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> |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 4.5.22 void CLOCK\_InitArmPll ( const ccm\_analog\_integer\_pll\_config\_t \* *config* )

Parameters

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

Note

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

#### 4.5.23 void CLOCK\_InitSysPll1 ( const ccm\_analog\_integer\_pll\_config\_t \* *config* )

Parameters

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

Note

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

#### 4.5.24 void CLOCK\_InitSysPll2 ( const ccm\_analog\_integer\_pll\_config\_t \* *config* )

Parameters

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

Note

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

#### 4.5.25 void CLOCK\_InitSysPll3 ( const ccm\_analog\_integer\_pll\_config\_t \* *config* )

## Parameters

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

## Note

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

#### 4.5.26 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.

#### 4.5.27 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.

#### 4.5.28 void CLOCK\_InitVideoPII1 ( 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). |
|---------------|-------------------------------------------------------------------------------------------------------|

#### 4.5.29 void CLOCK\_InitIntegerPll ( CCM\_ANALOG\_Type \* *base*, const ccm\_analog\_integer\_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_integer_pll_config_t</a> enumeration). |
| <i>type</i>   | integer pll type                                                                                         |

#### 4.5.30 uint32\_t CLOCK\_GetIntegerPllFreq ( 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>       | integer pll type           |
| <i>pll1Bypass</i> | pll1 bypass flag           |
| <i>refClkFreq</i> | Reference clock frequency. |

## Returns

Clock frequency

#### 4.5.31 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.                                                                                  |

#### 4.5.32 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>refClkFreq</i> | Reference clock frequency. |

## Returns

Clock frequency

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

## Parameters

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

## Returns

Clock frequency

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

## Parameters

|             |                  |
|-------------|------------------|
| <i>ctrl</i> | The pll control. |
|-------------|------------------|

## Returns

Clock frequency

#### 4.5.35 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 <code>clock_name_t</code> |
|------------------|--------------------------------------------------|

## Returns

Clock frequency value in hertz

#### 4.5.36 uint32\_t CLOCK\_GetClockRootFreq ( clock\_root\_t *clockRoot* )

## Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>clockRoot</i> | The clock root used to get the frequency, please refer to <a href="#">clock_root_t</a> . |
|------------------|------------------------------------------------------------------------------------------|

## Returns

The frequency of selected clock root.

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

## Returns

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

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

Returns

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

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

Returns

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

#### 4.5.40 uint32\_t CLOCK\_GetEnetAxiFreq ( void )

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

## Chapter 5

# IOMUXC: IOMUX Controller

### 5.1 Overview

IOMUXC driver provides APIs for pin configuration. It also supports the miscellaneous functions integrated in IOMUXC.

#### Files

- file [fsl\\_iomuxc.h](#)

#### Driver version

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

#### Pin function ID

The pin function ID is a tuple of <muxRegister muxMode inputRegister inputDaisy configRegister>

- #define **IOMUXC\_PMIC\_STBY\_REQ** 0x30330014, 0x0, 0x00000000, 0x0, 0x3033027C
- #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
- #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\_USDHC3\_RESET\_B** 0x3033004C, 0x4, 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\_USDHC3\_VSELECT** 0x30330054, 0x4, 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\_USDHC3\_CD\_B** 0x30330060, 0x4, 0x30330544, 0x2, 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\_USDHC3\_WP** 0x30330064, 0x4, 0x30330548, 0x2, 0x303302-CC
- #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
- #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, 0x3033031C
- #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, 0x3033032C
- #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, 0x3033033C
- #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, 0x3033034C
- #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\_DATA3\_SRC\_EARLY\_RESET** 0x303300E8, 0x6, 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\_RESET\_B\_SRC\_SYSTEM\_RESET** 0x303300EC, 0x6, 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, 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\_USDHC3\_STROBE** 0x303300FC, 0x2, 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\_USDHC3\_DATA5** 0x30330100, 0x2, 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\_USDHC3\_DATA6** 0x30330104, 0x2, 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\_USDHC3\_DATA7** 0x30330108, 0x2, 0x00000000, 0x0, 0x30330370

```

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_USDHC3_CD_B 0x30330114, 0x2, 0x30330544, 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_USDHC3_WP 0x30330118, 0x2, 0x30330548, 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_USDHC3_DATA0 0x3033011C, 0x2, 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_USDHC3_DATA1 0x30330120, 0x2, 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\_USDHC3\_DATA2** 0x30330124, 0x2, 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\_USDHC3\_DATA3** 0x30330128, 0x2, 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\_USDHC3\_DATA4** 0x30330130, 0x2, 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\_USDHC3\_RESET\_B** 0x30330134, 0x2, 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\_USDHC3\_CLK** 0x30330138, 0x2, 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
- #define **IOMUXC\_NAND\_WP\_B\_USDHC3\_CMD** 0x3033013C, 0x2, 0x00000000, 0x0, 0x303303A4
- #define **IOMUXC\_NAND\_WP\_B\_GPIO3\_IO18** 0x3033013C, 0x5, 0x00000000, 0x0, 0x303303A4
- #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\_PDM\_CLK** 0x30330144, 0x4, 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\_PDM\_BIT\_STREAM0** 0x30330148, 0x4, 0x30330534, 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\_PDM\_BIT\_STREAM1** 0x3033014C, 0x4, 0x30330538, 0x0, 0x303303B4
- #define **IOMUXC\_SAI5\_RXD1\_GPIO3\_IO22** 0x3033014C, 0x5, 0x00000000, 0x0, 0x303303B4
- #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\_PDM\_BIT\_STREAM2** 0x30330150, 0x4, 0x3033053C, 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\_PDM\_BIT\_STREAM3** 0x30330154, 0x4, 0x30330540, 0x0, 0x303303BC
- #define **IOMUXC\_SAI5\_RXD3\_GPIO3\_IO24** 0x30330154, 0x5, 0x00000000, 0x0, 0x303303BC
- #define **IOMUXC\_SAI5\_MCLK\_SAI5\_MCLK** 0x30330158, 0x0, 0x3033052C, 0x0, 0x303303C0
- #define **IOMUXC\_SAI5\_MCLK\_SAI1\_TX\_BCLK** 0x30330158, 0x1, 0x303304C8, 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\_SAI1\_TX\_DATA1** 0x30330164, 0x2, 0x00000000, 0x0, 0x303303CC
- #define **IOMUXC\_SAI1\_RXD0\_PDM\_BIT\_STREAM0** 0x30330164, 0x3, 0x30330534, 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\_PDM\_BIT\_STREAM1** 0x30330168, 0x3, 0x30330538, 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\_PDM\_BIT\_STREAM2** 0x3033016C, 0x3, 0x3033053C, 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
- #define **IOMUXC\_SAI1\_RXD3\_SAI5\_RX\_DATA3** 0x30330170, 0x1, 0x303304E0, 0x1, 0x303303D8
- #define **IOMUXC\_SAI1\_RXD3\_PDM\_BIT\_STREAM3** 0x30330170, 0x3, 0x30330540, 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
• #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
- #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, 0x3033040C
- #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\_PDM\_CLK** 0x303301A8, 0x3, 0x00000000, 0x0, 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\_PDM\_CLK** 0x303301AC, 0x3, 0x00000000, 0x0, 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\_SAI5\_TX\_DATA1** 0x303301B0, 0x2, 0x00000000, 0x0, 0x30330418
- #define **IOMUXC\_SAI2\_RXFS\_SAI2\_RX\_DATA1** 0x303301B0, 0x3, 0x00000000, 0x0, 0x30330418
- #define **IOMUXC\_SAI2\_RXFS\_UART1\_TX** 0x303301B0, 0x4, 0x00000000, 0x0, 0x30330418
- #define **IOMUXC\_SAI2\_RXFS\_UART1\_RX** 0x303301B0, 0x4, 0x303304F4, 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\_UART1\_RX** 0x303301B4, 0x4, 0x303304F4, 0x3, 0x3033041C
- #define **IOMUXC\_SAI2\_RXC\_UART1\_TX** 0x303301B4, 0x4, 0x00000000, 0x0, 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\_UART1\_RTS\_B** 0x303301B8, 0x4, 0x303304F0, 0x2, 0x30330420
- #define **IOMUXC\_SAI2\_RXD0\_UART1\_CTS\_B** 0x303301B8, 0x4, 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\_SAI2\_TX\_DATA1** 0x303301BC, 0x3, 0x00000000, 0x0, 0x30330424
- #define **IOMUXC\_SAI2\_TXFS\_UART1\_CTS\_B** 0x303301BC, 0x4, 0x00000000, 0x0, 0x30330424
- #define **IOMUXC\_SAI2\_TXFS\_UART1\_RTS\_B** 0x303301BC, 0x4, 0x303304F0, 0x3, 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, 0x3033042C
- #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\_SAI3\_RX\_DATA1** 0x303301CC, 0x3, 0x00000000, 0x0, 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\_CLK** 0x303301D0, 0x1, 0x00000000, 0x0, 0x30330438
- #define **IOMUXC\_SAI3\_RXC\_SAI5\_RX\_BCLK** 0x303301D0, 0x2, 0x303304D0, 0x2, 0x30330438
- #define **IOMUXC\_SAI3\_RXC\_UART2\_CTS\_B** 0x303301D0, 0x4, 0x00000000, 0x0, 0x30330438
- #define **IOMUXC\_SAI3\_RXC\_UART2\_RTS\_B** 0x303301D0, 0x4, 0x303304F8, 0x2, 0x30330438
- #define **IOMUXC\_SAI3\_RXC\_GPIO4\_IO29** 0x303301D0, 0x5, 0x00000000, 0x0, 0x30330438
- #define **IOMUXC\_SAI3\_RXD\_SAI3\_RX\_DATA0** 0x303301D4, 0x0, 0x00000000, 0x0, 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\_UART2\_RTS\_B** 0x303301D4, 0x4, 0x303304F8, 0x3, 0x3033043C
- #define **IOMUXC\_SAI3\_RXD\_UART2\_CTS\_B** 0x303301D4, 0x4, 0x00000000, 0x0, 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\_CAPTURE2** 0x303301D8, 0x1, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_SAI5\_RX\_DATA1** 0x303301D8, 0x2, 0x303304D8, 0x2, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_SAI3\_TX\_DATA1** 0x303301D8, 0x3, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_UART2\_RX** 0x303301D8, 0x4, 0x303304FC, 0x2, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_UART2\_TX** 0x303301D8, 0x4, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC\_SAI3\_TXFS\_GPIO4\_IO31** 0x303301D8, 0x5, 0x00000000, 0x0, 0x30330440
- #define **IOMUXC\_SAI3\_TXC\_SAI3\_TX\_BCLK** 0x303301DC, 0x0, 0x00000000, 0x0, 0x30330440

- 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\_UART2\_TX** 0x303301DC, 0x4, 0x00000000, 0x0, 0x30330444
- #define **IOMUXC\_SAI3\_TXC\_UART2\_RX** 0x303301DC, 0x4, 0x303304FC, 0x3, 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, 0x3033044-C
- #define **IOMUXC\_SAI3\_MCLK\_PWM4\_OUT** 0x303301E4, 0x1, 0x00000000, 0x0, 0x3033044-C
- #define **IOMUXC\_SAI3\_MCLK\_SAI5\_MCLK** 0x303301E4, 0x2, 0x3033052C, 0x3, 0x3033044-C
- #define **IOMUXC\_SAI3\_MCLK\_GPIO5\_IO02** 0x303301E4, 0x5, 0x00000000, 0x0, 0x3033044-C
- #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, 0x3033045-C
- #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
- #define **IOMUXC\_ECSP12\_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\_USDHC3\_CD\_B** 0x3033021C, 0x2, 0x30330544, 0x1, 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\_USDHC3\_WP** 0x30330220, 0x2, 0x30330548, 0x1, 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\_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, 0x303304-A4
- #define **IOMUXC\_UART2\_TXD\_UART2\_TX** 0x30330240, 0x0, 0x00000000, 0x0, 0x303304-A8
- #define **IOMUXC\_UART2\_TXD\_UART2\_RX** 0x30330240, 0x0, 0x303304FC, 0x1, 0x303304-A8
- #define **IOMUXC\_UART2\_TXD\_ECSPi3\_SS0** 0x30330240, 0x1, 0x00000000, 0x0, 0x303304-A8

- #define **IOMUXC\_UART2\_TXD\_GPIO5\_IO25** 0x30330240, 0x5, 0x00000000, 0x0, 0x303304-A8
- #define **IOMUXC\_UART3\_RXD\_UART3\_RX** 0x30330244, 0x0, 0x30330504, 0x2, 0x303304-AC
- #define **IOMUXC\_UART3\_RXD\_UART3\_TX** 0x30330244, 0x0, 0x00000000, 0x0, 0x303304-AC
- #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\_USDHC3\_RESET\_B** 0x30330244, 0x2, 0x00000000, 0x0, 0x303304AC
- #define **IOMUXC\_UART3\_RXD\_GPIO5\_IO26** 0x30330244, 0x5, 0x00000000, 0x0, 0x303304-AC
- #define **IOMUXC\_UART3\_TXD\_UART3\_TX** 0x30330248, 0x0, 0x00000000, 0x0, 0x303304-B0
- #define **IOMUXC\_UART3\_TXD\_UART3\_RX** 0x30330248, 0x0, 0x30330504, 0x3, 0x303304-B0
- #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\_USDHC3\_VSELECT** 0x30330248, 0x2, 0x00000000, 0x0, 0x303304B0
- #define **IOMUXC\_UART3\_TXD\_GPIO5\_IO27** 0x30330248, 0x5, 0x00000000, 0x0, 0x303304-B0
- #define **IOMUXC\_UART4\_RXD\_UART4\_RX** 0x3033024C, 0x0, 0x3033050C, 0x2, 0x303304-B4
- #define **IOMUXC\_UART4\_RXD\_UART4\_TX** 0x3033024C, 0x0, 0x00000000, 0x0, 0x303304-B4
- #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, 0x303304-B4
- #define **IOMUXC\_UART4\_TXD\_UART4\_TX** 0x30330250, 0x0, 0x00000000, 0x0, 0x303304-B8
- #define **IOMUXC\_UART4\_TXD\_UART4\_RX** 0x30330250, 0x0, 0x3033050C, 0x3, 0x303304-B8
- #define **IOMUXC\_UART4\_TXD\_UART2\_RTS\_B** 0x30330250, 0x1, 0x303304F8, 0x1, 0x303304B8
- #define **IOMUXC\_UART4\_TXD\_UART2\_CTS\_B** 0x30330250, 0x1, 0x00000000, 0x0, 0x303304B8
- #define **IOMUXC\_UART4\_TXD\_GPIO5\_IO29** 0x30330250, 0x5, 0x00000000, 0x0, 0x303304-B8
- #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** (uintptr\_t muxRegister, uint32\_t muxMode, uintptr\_t inputRegister, uint32\_t inputDaisy, uintptr\_t configRegister, uint32\_t inputOnfield)  
*Sets the IOMUXC pin mux mode.*
- static void **IOMUXC\_SetPinConfig** (uintptr\_t muxRegister, uint32\_t muxMode, uintptr\_t inputRegister, uint32\_t inputDaisy, uintptr\_t configRegister, uint32\_t configValue)  
*Sets the IOMUXC pin configuration.*

## 5.2 Macro Definition Documentation

### 5.2.1 #define FSL\_IOMUXC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## 5.3 Function Documentation

### 5.3.1 static void IOMUXC\_SetPinMux ( uintptr\_t muxRegister, uint32\_t muxMode, uintptr\_t inputRegister, uint32\_t inputDaisy, uintptr\_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>inputOnfield</i>   | The pad->module input inversion_ |

### 5.3.2 static void IOMUXC\_SetPinConfig ( uintptr\_t *muxRegister*, uint32\_t *muxMode*, uintptr\_t *inputRegister*, uint32\_t *inputDaisy*, uintptr\_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 6

## Common Driver

### 6.1 Overview

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

#### Macros

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

#### Typedefs

- typedef int32\_t `status_t`  
*Type used for all status and error return values.*

## Enumerations

- enum `_status_groups` {
  - `kStatusGroup_Generic` = 0,
  - `kStatusGroup_FLASH` = 1,
  - `kStatusGroup_LPSPI` = 4,
  - `kStatusGroup_FLEXIO_SPI` = 5,
  - `kStatusGroup_DSPI` = 6,
  - `kStatusGroup_FLEXIO_UART` = 7,
  - `kStatusGroup_FLEXIO_I2C` = 8,
  - `kStatusGroup_LPI2C` = 9,
  - `kStatusGroup_UART` = 10,
  - `kStatusGroup_I2C` = 11,
  - `kStatusGroup_LPSCI` = 12,
  - `kStatusGroup_LPUART` = 13,
  - `kStatusGroup_SPI` = 14,
  - `kStatusGroup_XRDC` = 15,
  - `kStatusGroup_SEMA42` = 16,
  - `kStatusGroup_SDHC` = 17,
  - `kStatusGroup_SDMMC` = 18,
  - `kStatusGroup_SAI` = 19,
  - `kStatusGroup_MCG` = 20,
  - `kStatusGroup_SCG` = 21,
  - `kStatusGroup_SDSPI` = 22,
  - `kStatusGroup_FLEXIO_I2S` = 23,
  - `kStatusGroup_FLEXIO_MCULCD` = 24,
  - `kStatusGroup_FLASHIAP` = 25,
  - `kStatusGroup_FLEXCOMM_I2C` = 26,
  - `kStatusGroup_I2S` = 27,
  - `kStatusGroup_IUART` = 28,
  - `kStatusGroup_CSI` = 29,
  - `kStatusGroup_MIPI_DSI` = 30,
  - `kStatusGroup_SDRAMC` = 35,
  - `kStatusGroup_POWER` = 39,
  - `kStatusGroup_ENET` = 40,
  - `kStatusGroup_PHY` = 41,
  - `kStatusGroup_TRGMUX` = 42,
  - `kStatusGroup_SMARTCARD` = 43,
  - `kStatusGroup_LMEM` = 44,
  - `kStatusGroup_QSPI` = 45,
  - `kStatusGroup_DMA` = 50,
  - `kStatusGroup_EDMA` = 51,
  - `kStatusGroup_DMAMGR` = 52,
  - `kStatusGroup_FLEXCAN` = 53,
  - `kStatusGroup_LTC` = 54,
  - `kStatusGroup_FLEXIO_CAMERA` = 55,
  - `kStatusGroup_LPC_SPI` = 56,
  - `kStatusGroup_LPC_USMCI` = 57,
  - `kStatusGroup_DMIC` = 58,
  - `kStatusGroup_SDIF` = 59,

```
kStatusGroup_NETC = 166 }
```

*Status group numbers.*

- enum {
  - kStatus\_Success = MAKE\_STATUS(kStatusGroup\_Generic, 0),
  - kStatus\_Fail = MAKE\_STATUS(kStatusGroup\_Generic, 1),
  - kStatus\_ReadOnly = MAKE\_STATUS(kStatusGroup\_Generic, 2),
  - kStatus\_OutOfRange = MAKE\_STATUS(kStatusGroup\_Generic, 3),
  - kStatus\_InvalidArgument = MAKE\_STATUS(kStatusGroup\_Generic, 4),
  - kStatus\_Timeout = MAKE\_STATUS(kStatusGroup\_Generic, 5),
  - kStatus\_NoTransferInProgress,
  - kStatus\_Busy = MAKE\_STATUS(kStatusGroup\_Generic, 7),
  - kStatus\_NoData }

*Generic status return codes.*

## Functions

- void \* **SDK\_Malloc** (size\_t size, size\_t alignbytes)
 

*Allocate memory with given alignment and aligned size.*
- void **SDK\_Free** (void \*ptr)
 

*Free memory.*
- void **SDK\_DelayAtLeastUs** (uint32\_t delayTime\_us, uint32\_t coreClock\_Hz)
 

*Delay at least for some time.*
- static status\_t **EnableIRQ** (IRQn\_Type interrupt)
 

*Enable specific interrupt.*
- static status\_t **DisableIRQ** (IRQn\_Type interrupt)
 

*Disable specific interrupt.*
- static status\_t **EnableIRQWithPriority** (IRQn\_Type interrupt, uint8\_t priNum)
 

*Enable the IRQ, and also set the interrupt priority.*
- static status\_t **IRQ\_SetPriority** (IRQn\_Type interrupt, uint8\_t priNum)
 

*Set the IRQ priority.*
- static status\_t **IRQ\_ClearPendingIRQ** (IRQn\_Type interrupt)
 

*Clear the pending IRQ flag.*
- static uint32\_t **DisableGlobalIRQ** (void)
 

*Disable the global IRQ.*
- static void **EnableGlobalIRQ** (uint32\_t primask)
 

*Enable the global IRQ.*

## Driver version

- #define **FSL\_COMMON\_DRIVER\_VERSION** (MAKE\_VERSION(2, 4, 0))
 

*common driver version.*

## Min/max macros

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

## UINT16\_MAX/UINT32\_MAX value

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

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

## Suppress fallthrough warning macro

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

## Atomic modification

These macros are used for atomic access, such as read-modify-write to the peripheral registers.

- **SDK\_ATOMIC\_LOCAL\_ADD**
- **SDK\_ATOMIC\_LOCAL\_SET**
- **SDK\_ATOMIC\_LOCAL\_CLEAR**
- **SDK\_ATOMIC\_LOCAL\_TOGGLE**
- **SDK\_ATOMIC\_LOCAL\_CLEAR\_AND\_SET**

Take **SDK\_ATOMIC\_LOCAL\_CLEAR\_AND\_SET** as an example: the parameter `addr` means the address of the peripheral register or variable you want to modify atomically, the parameter `clearBits` is the bits to clear, the parameter `setBits` it the bits to set. For example, to set a 32-bit register bit1:bit0 to 0b10, use like this:

```
volatile uint32_t * reg = (volatile uint32_t *)REG_ADDR;

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(reg, 0x03, 0x02);
```

In this example, the register bit1:bit0 are cleared and bit1 is set, as a result, register bit1:bit0 = 0b10.

### Note

For the platforms don't support exclusive load and store, these macros disable the global interrupt to preprotect the modification.

These macros only guarantee the local processor atomic operations. For the multi-processor devices, use hardware semaphore such as SEMA42 to guarantee exclusive access if necessary.

- #define **SDK\_ATOMIC\_LOCAL\_ADD**(addr, val)
- #define **SDK\_ATOMIC\_LOCAL\_SUB**(addr, val)
- #define **SDK\_ATOMIC\_LOCAL\_SET**(addr, bits)
- #define **SDK\_ATOMIC\_LOCAL\_CLEAR**(addr, bits)
- #define **SDK\_ATOMIC\_LOCAL\_TOGGLE**(addr, bits)
- #define **SDK\_ATOMIC\_LOCAL\_CLEAR\_AND\_SET**(addr, clearBits, setBits)

## Timer utilities

- #define **USEC\_TO\_COUNT**(us, clockFreqInHz) (uint64\_t)((((uint64\_t)(us) \* (clockFreqInHz)) / 1000000U))  
*Macro to convert a microsecond period to raw count value.*
- #define **COUNT\_TO\_USEC**(count, clockFreqInHz) (uint64\_t)((uint64\_t)(count)\*1000000U / (clockFreqInHz))  
*Macro to convert a raw count value to microsecond.*

- #define **MSEC\_TO\_COUNT**(ms, clockFreqInHz) (uint64\_t)((uint64\_t)(ms) \* (clockFreqInHz) / 1000U)  
*Macro to convert a millisecond period to raw count value.*
- #define **COUNT\_TO\_MSEC**(count, clockFreqInHz) (uint64\_t)((uint64\_t)(count)\*1000U / (clockFreqInHz))  
*Macro to convert a raw count value to millisecond.*

## Alignment variable definition macros

- #define **SDK\_SIZEALIGN**(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))  
*Macro to define a variable with L1 d-cache line size alignment.*

## Non-cacheable region definition macros

- #define **AT\_NONCACHEABLE\_SECTION**(var) var
- #define **AT\_NONCACHEABLE\_SECTION\_ALIGN**(var, alignbytes) SDK\_ALIGN(var, alignbytes)
- #define **AT\_NONCACHEABLE\_SECTION\_INIT**(var) var
- #define **AT\_NONCACHEABLE\_SECTION\_ALIGN\_INIT**(var, alignbytes) SDK\_ALIGN(var, alignbytes)

## 6.2 Macro Definition Documentation

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

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

### 6.2.3 #define MAKE\_VERSION( major, minor, bugfix ) (((major)\*65536L) + ((minor)\*256L) + (bugfix))

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

|        |               |               |         |   |
|--------|---------------|---------------|---------|---|
| Unused | Major Version | Minor Version | Bug Fix |   |
| 31     | 25 24         | 17 16         | 9 8     | 0 |

**6.2.4 #define FSL\_COMMON\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))**

**6.2.5 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE 0U**

**6.2.6 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART 1U**

**6.2.7 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART 2U**

**6.2.8 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI 3U**

**6.2.9 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC 4U**

**6.2.10 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM 5U**

**6.2.11 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART 6U**

**6.2.12 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART 7U**

**6.2.13 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART 8U**

**6.2.14 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO 9U**

**6.2.15 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI 10U**

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

**6.2.17 #define SDK\_SIZEALIGN( var, alignbytes ) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))**

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

## **6.3 Typedef Documentation**

**6.3.1 typedef int32\_t status\_t**

## 6.4 Enumeration Type Documentation

### 6.4.1 enum \_status\_groups

Enumerator

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



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

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

## 6.4.2 anonymous enum

Enumerator

*kStatus\_Success* Generic status for Success.  
*kStatus\_Fail* Generic status for Fail.  
*kStatus\_ReadOnly* Generic status for read only failure.  
*kStatus\_OutOfRange* Generic status for out of range access.

***kStatus\_InvalidArgument*** Generic status for invalid argument check.

***kStatus\_Timeout*** Generic status for timeout.

***kStatus\_NoTransferInProgress*** Generic status for no transfer in progress.

***kStatus\_Busy*** Generic status for module is busy.

***kStatus\_NoData*** Generic status for no data is found for the operation.

## 6.5 Function Documentation

### 6.5.1 void\* SDK\_Malloc ( size\_t size, size\_t alignbytes )

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>size</i>       | The length required to malloc. |
| <i>alignbytes</i> | The alignment size.            |

Return values

|            |                   |
|------------|-------------------|
| <i>The</i> | allocated memory. |
|------------|-------------------|

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

Parameters

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

### 6.5.3 void SDK\_DelayAtLeastUs ( uint32\_t delayTime\_us, uint32\_t coreClock\_Hz )

Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

|                     |                                    |
|---------------------|------------------------------------|
| <i>delayTime_us</i> | Delay time in unit of microsecond. |
| <i>coreClock_Hz</i> | Core clock frequency with Hz.      |

### 6.5.4 static status\_t EnableIRQ ( IRQn\_Type interrupt ) [inline], [static]

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they

are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

Parameters

|                  |                 |
|------------------|-----------------|
| <i>interrupt</i> | The IRQ number. |
|------------------|-----------------|

Return values

|                        |                                |
|------------------------|--------------------------------|
| <i>kStatus_Success</i> | Interrupt enabled successfully |
| <i>kStatus_Fail</i>    | Failed to enable the interrupt |

### 6.5.5 static status\_t DisableIRQ ( IRQn\_Type *interrupt* ) [inline], [static]

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

Parameters

|                  |                 |
|------------------|-----------------|
| <i>interrupt</i> | The IRQ number. |
|------------------|-----------------|

Return values

|                        |                                 |
|------------------------|---------------------------------|
| <i>kStatus_Success</i> | Interrupt disabled successfully |
| <i>kStatus_Fail</i>    | Failed to disable the interrupt |

### 6.5.6 static status\_t EnableIRQWithPriority ( IRQn\_Type *interrupt*, uint8\_t *priNum* ) [inline], [static]

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

## Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>interrupt</i> | The IRQ to Enable.                                    |
| <i>priNum</i>    | Priority number set to interrupt controller register. |

## Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Interrupt priority set successfully   |
| <i>kStatus_Fail</i>    | Failed to set the interrupt priority. |

### 6.5.7 static status\_t IRQ\_SetPriority ( IRQn\_Type *interrupt*, uint8\_t *priNum* ) [inline], [static]

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

## Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>interrupt</i> | The IRQ to set.                                       |
| <i>priNum</i>    | Priority number set to interrupt controller register. |

## Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Interrupt priority set successfully   |
| <i>kStatus_Fail</i>    | Failed to set the interrupt priority. |

### 6.5.8 static status\_t IRQ\_ClearPendingIRQ ( IRQn\_Type *interrupt* ) [inline], [static]

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

## Parameters

|                  |                              |
|------------------|------------------------------|
| <i>interrupt</i> | The flag which IRQ to clear. |
|------------------|------------------------------|

## Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Interrupt priority set successfully   |
| <i>kStatus_Fail</i>    | Failed to set the interrupt priority. |

### 6.5.9 static uint32\_t DisableGlobalIRQ ( void ) [inline], [static]

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the [EnableGlobalIRQ\(\)](#).

## Returns

Current primask value.

### 6.5.10 static void EnableGlobalIRQ ( uint32\_t primask ) [inline], [static]

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the [EnableGlobalIRQ\(\)](#) and [DisableGlobalIRQ\(\)](#) in pair.

## Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>primask</i> | value of primask register to be restored. The primask value is supposed to be provided by the <a href="#">DisableGlobalIRQ()</a> . |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|



## Chapter 7

# ECSPi: Enhanced Configurable Serial Peripheral Interface Driver

### 7.1 Overview

#### Modules

- [ECSPi CMSIS Driver](#)
- [ECSPi Driver](#)
- [ECSPi FreeRTOS Driver](#)
- [ECSPi SDMA Driver](#)

## 7.2 ECSPI Driver

### 7.2.1 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.

### 7.2.2 Typical use case

#### 7.2.2.1 SPI master transfer using polling method

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

#### 7.2.2.2 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` (0x00U)  
*ECSPI dummy transfer data, the data is sent while txBuff is NULL.*
- #define `SPI_RETRY_TIMES` 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*

## Typedefs

- typedef `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 {  
  `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),  
  `kStatus_ECSPI_Timeout` = MAKE\_STATUS(kStatusGroup\_ECSPI, 4) }  
*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 {  
  `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 {  
`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 {  
`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_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, 3, 2))  
*ECSPI driver version.*

## Initialization and deinitialization

- `void ECSPI_MasterGetDefaultConfig` (ecspi\_master\_config\_t \*config)  
*Sets the ECSPI configuration structure to default values.*
- `void ECSPI_MasterInit` (ECSPI\_Type \*base, const ecspi\_master\_config\_t \*config, uint32\_t src-Clock\_Hz)  
*Initializes the ECSPI with configuration.*
- `void ECSPI_SlaveGetDefaultConfig` (ecspi\_slave\_config\_t \*config)  
*Sets the ECSPI configuration structure to default values.*
- `void ECSPI_SlaveInit` (ECSPI\_Type \*base, const ecspi\_slave\_config\_t \*config)  
*Initializes the ECSPI with configuration.*
- `void ECSPI_Deinit` (ECSPI\_Type \*base)  
*De-initializes the ECSPI.*
- `static void ECSPI_Enable` (ECSPI\_Type \*base, bool enable)  
*Enables or disables the ECSPI.*

## Status

- `static uint32_t ECSPI_GetStatusFlags` (ECSPI\_Type \*base)  
*Gets the status flag.*
- `static void ECSPI_ClearStatusFlags` (ECSPI\_Type \*base, uint32\_t mask)  
*Clear the status flag.*

## Interrupts

- `static void ECSPI_EnableInterrupts` (ECSPI\_Type \*base, uint32\_t mask)  
*Enables the interrupt for the ECSPI.*
- `static void ECSPI_DisableInterrupts` (ECSPI\_Type \*base, uint32\_t mask)  
*Disables the interrupt for the ECSPI.*

## Software Reset

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

## Channel mode check

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

## DMA Control

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

## FIFO Operation

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

## Bus Operations

- static void [ECSPI\\_SetChannelSelect](#) (ECSPI\_Type \*base, [ecspi\\_channel\\_source\\_t](#) channel)  
*Set channel select for transfer.*
- void [ECSPI\\_SetChannelConfig](#) (ECSPI\_Type \*base, [ecspi\\_channel\\_source\\_t](#) channel, const [ecspi\\_channel\\_config\\_t](#) \*config)  
*Set channel select configuration for transfer.*
- void [ECSPI\\_SetBaudRate](#) (ECSPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the baud rate for ECSPI transfer.*
- [status\\_t](#) [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.*

## 7.2.3 Data Structure Documentation

### 7.2.3.1 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\\_clock\\_polarity\\_t](#) polarity  
*Clock polarity.*
- [ecspi\\_clock\\_phase\\_t](#) phase  
*Clock phase.*

### 7.2.3.2 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.*
- [uint16\\_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.*

#### Field Documentation

##### (1) [uint16\\_t ecspi\\_master\\_config\\_t::burstLength](#)

The length shall be less than 4096 bits

##### (2) [bool ecspi\\_master\\_config\\_t::enableLoopback](#)

### 7.2.3.3 struct ecspi\_slave\_config\_t

#### Data Fields

- [uint16\\_t burstLength](#)  
*Burst length.*
- [uint8\\_t txFifoThreshold](#)  
*TX Threshold.*
- [uint8\\_t rxFifoThreshold](#)  
*RX Threshold.*
- [ecspi\\_channel\\_config\\_t channelConfig](#)  
*Channel configuration.*

#### Field Documentation

##### (1) [uint16\\_t ecspi\\_slave\\_config\\_t::burstLength](#)

The length shall be less than 4096 bits

### 7.2.3.4 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.*

### 7.2.3.5 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.*

## 7.2.4 Macro Definition Documentation

**7.2.4.1 #define FSL\_ECSPi\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 2))**

**7.2.4.2 #define ECSPI\_DUMMYDATA (0x00U)**

**7.2.4.3 #define SPI\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

## 7.2.5 Enumeration Type Documentation

### 7.2.5.1 anonymous enum

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.  
*kStatus\_ECSPi\_Timeout* ECSPI timeout polling status flags.

### 7.2.5.2 enum ecspi\_clock\_polarity\_t

Enumerator

*kECSPi\_PolarityActiveHigh* Active-high ECSPi polarity high (idles low).  
*kECSPi\_PolarityActiveLow* Active-low ECSPi polarity low (idles high).

### 7.2.5.3 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.

### 7.2.5.4 anonymous enum

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.

#### 7.2.5.5 anonymous enum

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.

#### 7.2.5.6 anonymous enum

Enumerator

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

#### 7.2.5.7 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.

#### 7.2.5.8 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.

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

Enumerator

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

### 7.2.5.10 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.

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

Enumerator

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

### 7.2.5.12 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.

### 7.2.5.13 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).

## 7.2.6 Function Documentation

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

## Parameters

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

### 7.2.6.2 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 |
|---------------|-----------------------------|

### 7.2.6.3 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.                   |

### 7.2.6.4 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 |
|---------------|-----------------------------|

### 7.2.6.5 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_slaveconfig_t config = {
 .baudRate_Bps = 400000,
 ...
};
ECSPI_SlaveInit(ECSPI1, &config);
```

## Parameters

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

### 7.2.6.6 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 |
|-------------|--------------------|

### 7.2.6.7 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 |

### 7.2.6.8 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.

#### 7.2.6.9 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 <code>_ecspi_flags</code> could get the related status. |

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

## Parameters

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ECSPI base pointer                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <i>mask</i> | <p>ECSPI interrupt source. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> <li>• <code>kECSPI_TxFifoEmptyInterruptEnable</code></li> <li>• <code>kECSPI_TxFifoDataRequestInterruptEnable</code></li> <li>• <code>kECSPI_TxFifoFullInterruptEnable</code></li> <li>• <code>kECSPI_RxFifoReadyInterruptEnable</code></li> <li>• <code>kECSPI_RxFifoDataRequestInterruptEnable</code></li> <li>• <code>kECSPI_RxFifoFullInterruptEnable</code></li> <li>• <code>kECSPI_RxFifoOverflowInterruptEnable</code></li> <li>• <code>kECSPI_TransferCompleteInterruptEnable</code></li> <li>• <code>kECSPI_AllInterruptEnable</code></li> </ul> |

#### 7.2.6.11 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> |

**7.2.6.12** `static void ECSPI_SoftwareReset ( ECSPI_Type * base ) [inline], [static]`

#### Parameters

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

**7.2.6.13** `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

**7.2.6.14** `static void ECSPI_EnableDMA ( ECSPI_Type * base, uint32_t mask, bool enable ) [inline], [static]`

## Parameters

|               |                                                                                                                                                                                                         |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | ECSPI base pointer                                                                                                                                                                                      |
| <i>mask</i>   | ECSPI DMA source. The parameter can be any of the following values: <ul style="list-style-type: none"> <li>• kECSPI_TxDmaEnable</li> <li>• kECSPI_RxDmaEnable</li> <li>• kECSPI_DmaAllEnable</li> </ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA                                                                                                                                                          |

### 7.2.6.15 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.

### 7.2.6.16 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.

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

## Parameters

|                |                    |
|----------------|--------------------|
| <i>base</i>    | ECSPI base pointer |
| <i>channel</i> | Channel source.    |

#### 7.2.6.18 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 |

#### 7.2.6.19 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. |

#### 7.2.6.20 *status\_t* 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 |

#### Return values

|                              |                                         |
|------------------------------|-----------------------------------------|
| <i>kStatus_Success</i>       | Successfully start a transfer.          |
| <i>kStatus_ECSPI_Timeout</i> | The transfer timed out and was aborted. |

**7.2.6.21 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. |

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

#### Parameters

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

#### Returns

Data in the register.

**7.2.6.23 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.             |
| <i>userData</i> | User data.                     |

#### 7.2.6.24 **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.              |
| <i>kStatus_ECSPITimeout</i>    | The transfer timed out and was aborted. |

#### 7.2.6.25 **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. |

#### 7.2.6.26 **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. |

#### 7.2.6.27 **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. |

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

Parameters

---

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

**7.2.6.29** `void ECSPISlaveTransferCreateHandle ( ECSPISlaveType * base,  
ecspi_slave_handle_t * handle, ecspi_slave_callback_t callback, void * userData  
)`

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

Parameters

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

**7.2.6.30** `static status_t ECSPISlaveTransferNonBlocking ( ECSPISlaveType * 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_ECSPISlave_Busy</i> | ECSPI is not idle, is running another transfer. |

**7.2.6.31** `static status_t ECSPISlaveTransferGetCount ( ECSPISlaveType * 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. |

**7.2.6.32 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. |

**7.2.6.33 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 |

## 7.3 ECSPI FreeRTOS Driver

### 7.3.1 Overview

#### Driver version

- #define `FSL_ECSPI_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 0)`)  
*ECSPI FreeRTOS driver version.*

### 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.*

### 7.3.2 Macro Definition Documentation

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

### 7.3.3 Function Documentation

#### 7.3.3.1 `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.

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

This function deinitializes the ECSPI module and related RTOS context.

## Parameters

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

### 7.3.3.3 `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.



## 7.4 ECSPI SDMA Driver

### 7.4.1 Overview

#### Data Structures

- struct [ecspi\\_sdma\\_handle\\_t](#)  
*ECSPI SDMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* [ecspi\\_sdma\\_callback\\_t](#))(ECSPI\_Type \*base, ecspi\_sdma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*ECSPI SDMA callback called at the end of transfer.*

#### Driver version

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

#### DMA Transactional

- void [ECSPI\\_MasterTransferCreateHandleSDMA](#) (ECSPI\_Type \*base, ecspi\_sdma\_handle\_t \*handle, [ecspi\\_sdma\\_callback\\_t](#) callback, void \*userData, [sdma\\_handle\\_t](#) \*txHandle, [sdma\\_handle\\_t](#) \*rxHandle, uint32\_t eventSourceTx, uint32\_t eventSourceRx, uint32\_t TxChannel, uint32\_t RxChannel)  
*Initialize the ECSPI master SDMA handle.*
- void [ECSPI\\_SlaveTransferCreateHandleSDMA](#) (ECSPI\_Type \*base, ecspi\_sdma\_handle\_t \*handle, [ecspi\\_sdma\\_callback\\_t](#) callback, void \*userData, [sdma\\_handle\\_t](#) \*txHandle, [sdma\\_handle\\_t](#) \*rxHandle, uint32\_t eventSourceTx, uint32\_t eventSourceRx, uint32\_t TxChannel, uint32\_t RxChannel)  
*Initialize the ECSPI slave SDMA handle.*
- [status\\_t](#) [ECSPI\\_MasterTransferSDMA](#) (ECSPI\_Type \*base, ecspi\_sdma\_handle\_t \*handle, [ecspi\\_transfer\\_t](#) \*xfer)  
*Perform a non-blocking ECSPI master transfer using SDMA.*
- [status\\_t](#) [ECSPI\\_SlaveTransferSDMA](#) (ECSPI\_Type \*base, ecspi\_sdma\_handle\_t \*handle, [ecspi\\_transfer\\_t](#) \*xfer)  
*Perform a non-blocking ECSPI slave transfer using SDMA.*
- void [ECSPI\\_MasterTransferAbortSDMA](#) (ECSPI\_Type \*base, ecspi\_sdma\_handle\_t \*handle)  
*Abort a ECSPI master transfer using SDMA.*
- void [ECSPI\\_SlaveTransferAbortSDMA](#) (ECSPI\_Type \*base, ecspi\_sdma\_handle\_t \*handle)  
*Abort a ECSPI slave transfer using SDMA.*

## 7.4.2 Data Structure Documentation

### 7.4.2.1 struct \_ecspi\_sdma\_handle

#### Data Fields

- bool [txInProgress](#)  
*Send transfer finished.*
- bool [rxInProgress](#)  
*Receive transfer finished.*
- [sdma\\_handle\\_t](#) \* [txSdmaHandle](#)  
*DMA handler for ECSPI send.*
- [sdma\\_handle\\_t](#) \* [rxSdmaHandle](#)  
*DMA handler for ECSPI receive.*
- [ecspi\\_sdma\\_callback\\_t](#) [callback](#)  
*Callback for ECSPI SDMA transfer.*
- void \* [userData](#)  
*User Data for ECSPI SDMA callback.*
- uint32\_t [state](#)  
*Internal state of ECSPI SDMA transfer.*
- uint32\_t [ChannelTx](#)  
*Channel for send handle.*
- uint32\_t [ChannelRx](#)  
*Channel for receive handler.*

## 7.4.3 Macro Definition Documentation

### 7.4.3.1 #define FSL\_ECSPi\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

## 7.4.4 Typedef Documentation

### 7.4.4.1 typedef void(\* ecspi\_sdma\_callback\_t)(ECSPi\_Type \*base, ecspi\_sdma\_handle\_t \*handle, status\_t status, void \*userData)

## 7.4.5 Function Documentation

### 7.4.5.1 void ECSPi\_MasterTransferCreateHandleSDMA ( ECSPi\_Type \* base, ecspi\_sdma\_handle\_t \* handle, ecspi\_sdma\_callback\_t callback, void \* userData, sdma\_handle\_t \* txHandle, sdma\_handle\_t \* rxHandle, uint32\_t eventSourceTx, uint32\_t eventSourceRx, uint32\_t TxChannel, uint32\_t RxChannel )

This function initializes the ECSPI master SDMA handle which can be used for other SPI master transactional APIs. Usually, for a specified ECSPI instance, user need only 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>      | User callback function called at the end of a transfer.                          |
| <i>userData</i>      | User data for callback.                                                          |
| <i>txHandle</i>      | SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users. |
| <i>rxHandle</i>      | SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users. |
| <i>eventSourceTx</i> | event source for ECSPI send, which can be found in SDMA mapping.                 |
| <i>eventSourceRx</i> | event source for ECSPI receive, which can be found in SDMA mapping.              |
| <i>TxChannel</i>     | SDMA channel for ECSPI send.                                                     |
| <i>RxChannel</i>     | SDMA channel for ECSPI receive.                                                  |

**7.4.5.2 void ECSPI\_SlaveTransferCreateHandleSDMA ( ECSPI\_Type \* *base*,  
 ecspi\_sdma\_handle\_t \* *handle*, ecspi\_sdma\_callback\_t *callback*, void \* *userData*,  
 sdma\_handle\_t \* *txHandle*, sdma\_handle\_t \* *rxHandle*, uint32\_t *eventSourceTx*,  
 uint32\_t *eventSourceRx*, uint32\_t *TxChannel*, uint32\_t *RxChannel* )**

This function initializes the ECSPI Slave SDMA handle which can be used for other SPI Slave transactional APIs. Usually, for a specified ECSPI instance, user need only 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>      | User callback function called at the end of a transfer.                          |
| <i>userData</i>      | User data for callback.                                                          |
| <i>txHandle</i>      | SDMA handle pointer for ECSPI Tx, the handle shall be static allocated by users. |
| <i>rxHandle</i>      | SDMA handle pointer for ECSPI Rx, the handle shall be static allocated by users. |
| <i>eventSourceTx</i> | event source for ECSPI send, which can be found in SDMA mapping.                 |
| <i>eventSourceRx</i> | event source for ECSPI receive, which can be found in SDMA mapping.              |

|                  |                                 |
|------------------|---------------------------------|
| <i>TxChannel</i> | SDMA channel for ECSPI send.    |
| <i>RxChannel</i> | SDMA channel for ECSPI receive. |

#### 7.4.5.3 **status\_t ECSPI\_MasterTransferSDMA ( ECSPI\_Type \* *base*, ecspi\_sdma\_handle\_t \* *handle*, ecspi\_transfer\_t \* *xfer* )**

Note

This interface returned immediately after transfer initiates.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | ECSPI peripheral base address.      |
| <i>handle</i> | ECSPI SDMA handle pointer.          |
| <i>xfer</i>   | Pointer to sdma transfer structure. |

Return values

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

#### 7.4.5.4 **status\_t ECSPI\_SlaveTransferSDMA ( ECSPI\_Type \* *base*, ecspi\_sdma\_handle\_t \* *handle*, ecspi\_transfer\_t \* *xfer* )**

Note

This interface returned immediately after transfer initiates.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | ECSPI peripheral base address.      |
| <i>handle</i> | ECSPI SDMA handle pointer.          |
| <i>xfer</i>   | Pointer to sdma transfer structure. |

Return values

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

#### 7.4.5.5 void ECSPI\_MasterTransferAbortSDMA ( ECSPI\_Type \* *base*, ecsapi\_sdma\_handle\_t \* *handle* )

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | ECSPI peripheral base address. |
| <i>handle</i> | ECSPI SDMA handle pointer.     |

#### 7.4.5.6 void ECSPI\_SlaveTransferAbortSDMA ( ECSPI\_Type \* *base*, ecsapi\_sdma\_handle\_t \* *handle* )

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | ECSPI peripheral base address. |
| <i>handle</i> | ECSPI SDMA handle pointer.     |

## 7.5 ECSPI CMSIS Driver

This section describes the programming interface of the ecspi Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

### 7.5.1 Function groups

#### 7.5.1.1 ECSPI CMSIS GetVersion Operation

This function group will return the ECSPI CMSIS Driver version to user.

#### 7.5.1.2 ECSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 7.5.1.3 ECSPI CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the instance in master mode or slave mode. And this API must be called before you configure an instance or after you Deinit an instance. The right steps to start an instance is that you must initialize the instance which been selected firstly, then you can power on the instance. After these all have been done, you can configure the instance by using control operation. If you want to Uninitialize the instance, you must power off the instance first.

#### 7.5.1.4 ECSPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

#### 7.5.1.5 ECSPI CMSIS Status Operation

This function group gets the ecspi transfer status.

#### 7.5.1.6 ECSPI CMSIS Control Operation

This function can select instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer, set transfer data bits and set other control command.

## 7.5.2 Typical use case

### 7.5.2.1 Master Operation

```

/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*ECSPI master init*/
Driver_SPI0.Initialize(ECSPI_MasterSignalEvent_t);
Driver_SPI0.PowerControl(ARM_POWER_FULL);
Driver_SPI0.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
Driver_SPI0.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
Driver_SPI0.PowerControl(ARM_POWER_OFF);

/* Master uninitialized */
Driver_SPI0.Uninitialize();

```

### 7.5.2.2 Slave Operation

```

/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*DSPI slave init*/
Driver_SPI2.Initialize(ECSPI_SlaveSignalEvent_t);
Driver_SPI2.PowerControl(ARM_POWER_FULL);
Driver_SPI2.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
Driver_SPI2.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
Driver_SPI2.PowerControl(ARM_POWER_OFF);

/* slave uninitialized */
Driver_SPI2.Uninitialize();

```

## Chapter 8

# ENET: Ethernet MAC Driver

### 8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 10/100 Mbps Ethernet MAC (ENET) module of MCUXpresso SDK devices.

### ENET: Ethernet MAC Driver {EthernetMACDriver}

### 8.2 Operations of Ethernet MAC Driver

#### 8.2.1 MII interface Operation

The MII interface is the interface connected with MAC and PHY. the Serial management interface - MII management interface should be set before any access to the external PHY chip register. Call [ENET\\_SetSMI\(\)](#) to initialize the MII management interface. Use [ENET\\_StartSMIRead\(\)](#), [ENET\\_StartSMIWrite\(\)](#), and [ENET\\_ReadSMIData\(\)](#) to read/write to PHY registers. This function group sets up the MII and serial management SMI interface, gets data from the SMI interface, and starts the SMI read and write command. Use [ENET\\_SetMII\(\)](#) to configure the MII before successfully getting data from the external PHY.

#### 8.2.2 MAC address filter

This group sets/gets the ENET mac address and the multicast group address filter. [ENET\\_AddMulticastGroup\(\)](#) should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

#### 8.2.3 Other Basic control Operations

This group has the receive active API [ENET\\_ActiveRead\(\)](#) for single and multiple rings. The [ENET\\_AVBConfigure\(\)](#) is provided to configure the AVB features to support the AVB frames transmission. Note that due to the AVB frames transmission scheme being a credit-based TX scheme, it is only supported with the Enhanced buffer descriptors. Because of this, the AVB configuration should only be done with the Enhanced buffer descriptor. When the AVB feature is required, make sure the "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" is defined before using this feature.

#### 8.2.4 Transactional Operation

For ENET receive, the [ENET\\_GetRxFrameSize\(\)](#) function needs to be called to get the received data size. Then, call the [ENET\\_ReadFrame\(\)](#) function to get the received data. If the received error occurs, call the



[ENET\\_GetRxErrBeforeReadFrame\(\)](#) function after [ENET\\_GetRxFrameSize\(\)](#) and before [ENET\\_ReadFrame\(\)](#) functions to get the detailed error information.

For ENET transmit, call the [ENET\\_SendFrame\(\)](#) function to send the data out. The transmit data error information is only accessible for the IEEE 1588 enhanced buffer descriptor mode. When the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined, the [ENET\\_GetTxErrAfterSendFrame\(\)](#) can be used to get the detail transmit error information. The transmit error information can only be updated by uDMA after the data is transmitted. The [ENET\\_GetTxErrAfterSendFrame\(\)](#) function is recommended to be called on the transmit interrupt handler.

If send/read frame with zero-copy mechanism is needed, there're special APIs like [ENET\\_GetRxFrame\(\)](#) and [ENET\\_StartTxFrame\(\)](#). The send frame zero-copy APIs can't be used mixed with [ENET\\_SendFrame\(\)](#) for the same ENET peripheral, same as read frame zero-copy APIs.

## 8.2.5 PTP IEEE 1588 Feature Operation

This function group configures the PTP IEEE 1588 feature, starts/stops/gets/sets/adjusts the PTP IEEE 1588 timer, gets the receive/transmit frame timestamp, and PTP IEEE 1588 timer channel feature setting.

The [ENET\\_Ptp1588Configure\(\)](#) function needs to be called when the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined and the IEEE 1588 feature is required.

## 8.3 Typical use case

### 8.3.1 ENET Initialization, receive, and transmit operations

For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` undefined use case, use the legacy type buffer descriptor transmit/receive the frame as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet` For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` defined use case, add the PTP IEEE 1588 configuration to enable the PTP IEEE 1588 feature. The initialization occurs as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet`

## Data Structures

- struct [enet\\_rx\\_bd\\_struct\\_t](#)  
*Defines the receive buffer descriptor structure for the little endian system. [More...](#)*
- struct [enet\\_tx\\_bd\\_struct\\_t](#)  
*Defines the enhanced transmit buffer descriptor structure for the little endian system. [More...](#)*
- struct [enet\\_data\\_error\\_stats\\_t](#)  
*Defines the ENET data error statistics structure. [More...](#)*
- struct [enet\\_rx\\_frame\\_error\\_t](#)  
*Defines the Rx frame error structure. [More...](#)*
- struct [enet\\_transfer\\_stats\\_t](#)  
*Defines the ENET transfer statistics structure. [More...](#)*
- struct [enet\\_frame\\_info\\_t](#)  
*Defines the frame info structure. [More...](#)*

- struct `enet_tx_dirty_ring_t`  
Defines the ENET transmit dirty addresses ring/queue structure. [More...](#)
- struct `enet_buffer_config_t`  
Defines the receive buffer descriptor configuration structure. [More...](#)
- struct `enet_config_t`  
Defines the basic configuration structure for the ENET device. [More...](#)
- struct `enet_tx_bd_ring_t`  
Defines the ENET transmit buffer descriptor ring/queue structure. [More...](#)
- struct `enet_rx_bd_ring_t`  
Defines the ENET receive buffer descriptor ring/queue structure. [More...](#)
- struct `enet_handle_t`  
Defines the ENET handler structure. [More...](#)

## Macros

- #define `ENET_BUFFDESCRIPTOR_RX_ERR_MASK`  
Defines the receive error status flag mask.

## Typedefs

- typedef void `*(enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)`  
Defines the ENET Rx memory buffer alloc function pointer.
- typedef void `*(enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)`  
Defines the ENET Rx memory buffer free function pointer.
- typedef void `*(enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)`  
ENET callback function.
- typedef void `*(enet_isr_t)(ENET_Type *base, enet_handle_t *handle)`  
Define interrupt IRQ handler.

## Enumerations

- enum {  
    `kStatus_ENET_InitMemoryFail`,  
    `kStatus_ENET_RxFrameError` = MAKE\_STATUS(kStatusGroup\_ENET, 1U),  
    `kStatus_ENET_RxFrameFail` = MAKE\_STATUS(kStatusGroup\_ENET, 2U),  
    `kStatus_ENET_RxFrameEmpty` = MAKE\_STATUS(kStatusGroup\_ENET, 3U),  
    `kStatus_ENET_RxFrameDrop` = MAKE\_STATUS(kStatusGroup\_ENET, 4U),  
    `kStatus_ENET_TxFrameOverLen` = MAKE\_STATUS(kStatusGroup\_ENET, 5U),  
    `kStatus_ENET_TxFrameBusy` = MAKE\_STATUS(kStatusGroup\_ENET, 6U),  
    `kStatus_ENET_TxFrameFail` = MAKE\_STATUS(kStatusGroup\_ENET, 7U) }  
    Defines the status return codes for transaction.
- enum `enet_mii_mode_t` {  
    `kENET_MiiMode` = 0U,  
    `kENET_RmiiMode` = 1U }  
    Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.
- enum `enet_mii_speed_t` {  
    `kENET_MiiSpeed10M` = 0U,

```
kENET_MiiSpeed100M = 1U }
```

*Defines the 10/100/1000 Mbps speed for the MII data interface.*

- enum `enet_mii_duplex_t` {  
`kENET_MiiHalfDuplex` = 0U,  
`kENET_MiiFullDuplex` }

*Defines the half or full duplex for the MII data interface.*

- enum `enet_mii_write_t` {  
`kENET_MiiWriteNoCompliant` = 0U,  
`kENET_MiiWriteValidFrame` }

*Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.*

- enum `enet_mii_read_t` {  
`kENET_MiiReadValidFrame` = 2U,  
`kENET_MiiReadNoCompliant` = 3U }

*Defines the read operation for the MII management frame.*

- enum `enet_special_control_flag_t` {  
`kENET_ControlFlowControlEnable` = 0x0001U,  
`kENET_ControlRxPayloadCheckEnable` = 0x0002U,  
`kENET_ControlRxPadRemoveEnable` = 0x0004U,  
`kENET_ControlRxBroadCastRejectEnable` = 0x0008U,  
`kENET_ControlMacAddrInsert` = 0x0010U,  
`kENET_ControlStoreAndFwdDisable` = 0x0020U,  
`kENET_ControlSMIPreambleDisable` = 0x0040U,  
`kENET_ControlPromiscuousEnable` = 0x0080U,  
`kENET_ControlMIILoopEnable` = 0x0100U,  
`kENET_ControlVLANTagEnable` = 0x0200U }

*Defines a special configuration for ENET MAC controller.*

- enum `enet_interrupt_enable_t` {  
`kENET_BabrInterrupt` = ENET\_EIR\_BABR\_MASK,  
`kENET_BabtInterrupt` = ENET\_EIR\_BABT\_MASK,  
`kENET_GraceStopInterrupt` = ENET\_EIR\_GRA\_MASK,  
`kENET_TxFrameInterrupt` = ENET\_EIR\_TXF\_MASK,  
`kENET_TxBufferInterrupt` = ENET\_EIR\_TXB\_MASK,  
`kENET_RxFrameInterrupt` = ENET\_EIR\_RXF\_MASK,  
`kENET_RxBufferInterrupt` = ENET\_EIR\_RXB\_MASK,  
`kENET_MiiInterrupt` = ENET\_EIR\_MII\_MASK,  
`kENET_EBusERInterrupt` = ENET\_EIR\_EBERR\_MASK,  
`kENET_LateCollisionInterrupt` = ENET\_EIR\_LC\_MASK,  
`kENET_RetryLimitInterrupt` = ENET\_EIR\_RL\_MASK,  
`kENET_UnderrunInterrupt` = ENET\_EIR\_UN\_MASK,  
`kENET_PayloadRxInterrupt` = ENET\_EIR\_PLR\_MASK,  
`kENET_WakeupInterrupt` = ENET\_EIR\_WAKEUP\_MASK,  
`kENET_TsAvailInterrupt` = ENET\_EIR\_TS\_AVAIL\_MASK,  
`kENET_TsTimerInterrupt` = ENET\_EIR\_TS\_TIMER\_MASK }

*List of interrupts supported by the peripheral.*

- enum `enet_event_t` {

```

kENET_RxEvent,
kENET_TxEvent,
kENET_ErrEvent,
kENET_WakeUpEvent,
kENET_TimeStampEvent,
kENET_TimeStampAvailEvent }

```

*Defines the common interrupt event for callback use.*

- enum `enet_tx_accelerator_t` {  
`kENET_TxAccelIsShift16Enabled` = `ENET_TACC_SHIFT16_MASK`,  
`kENET_TxAccelIpCheckEnabled` = `ENET_TACC_IPCHK_MASK`,  
`kENET_TxAccelProtoCheckEnabled` = `ENET_TACC_PROCHK_MASK` }

*Defines the transmit accelerator configuration.*

- enum `enet_rx_accelerator_t` {  
`kENET_RxAccelPadRemoveEnabled` = `ENET_RACC_PADREM_MASK`,  
`kENET_RxAccelIpCheckEnabled` = `ENET_RACC_IPDIS_MASK`,  
`kENET_RxAccelProtoCheckEnabled` = `ENET_RACC_PRODIS_MASK`,  
`kENET_RxAccelMacCheckEnabled` = `ENET_RACC_LINEDIS_MASK`,  
`kENET_RxAccelIsShift16Enabled` = `ENET_RACC_SHIFT16_MASK` }

*Defines the receive accelerator configuration.*

## Functions

- `uint32_t ENET_GetInstance` (`ENET_Type *base`)  
*Get the ENET instance from peripheral base address.*

## Variables

- `const clock_ip_name_t s_enetClock []`  
*Pointers to enet clocks for each instance.*

## Driver version

- `#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 7, 0))`  
*Defines the driver version.*

## ENET DESCRIPTOR QUEUE

- `#define FSL_FEATURE_ENET_QUEUE 1` /\* Singal queue for previous IP. \*/  
*Defines the queue number.*

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

- `#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U`  
*Empty bit mask.*
- `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U`  
*Software owner one mask.*
- `#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U`  
*Next buffer descriptor is the start address.*

- #define `ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask` 0x1000U  
*Software owner two mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_LAST_MASK` 0x0800U  
*Last BD of the frame mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_MISS_MASK` 0x0100U  
*Received because of the promiscuous mode.*
- #define `ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK` 0x0080U  
*Broadcast packet mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK` 0x0040U  
*Multicast packet mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK` 0x0020U  
*Length violation mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK` 0x0010U  
*Non-octet aligned frame mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_CRC_MASK` 0x0004U  
*CRC error mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK` 0x0002U  
*FIFO overrun mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK` 0x0001U  
*Frame is truncated mask.*

### Control and status bit masks of the transmit buffer descriptor.

- #define `ENET_BUFFDESCRIPTOR_TX_READY_MASK` 0x8000U  
*Ready bit mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK` 0x4000U  
*Software owner one mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_WRAP_MASK` 0x2000U  
*Wrap buffer descriptor mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK` 0x1000U  
*Software owner two mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_LAST_MASK` 0x0800U  
*Last BD of the frame mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_TRANSMITCRC_MASK` 0x0400U  
*Transmit CRC mask.*

### Defines some Ethernet parameters.

- #define `ENET_FRAME_MAX_FRAMELEN` 1518U  
*Default maximum Ethernet frame size without VLAN tag.*
- #define `ENET_FRAME_VLAN_TAGLEN` 4U  
*Ethernet single VLAN tag size.*
- #define `ENET_FRAME_CRC_LEN` 4U  
*CRC size in a frame.*
- #define `ENET_FRAME_TX_LEN_LIMITATION(x)` (((x)->RCR & ENET\_RCR\_MAX\_FL\_MASK) >> ENET\_RCR\_MAX\_FL\_SHIFT) - `ENET_FRAME_CRC_LEN`
- #define `ENET_FIFO_MIN_RX_FULL` 5U  
*ENET minimum receive FIFO full.*
- #define `ENET_RX_MIN_BUFFERSIZE` 256U  
*ENET minimum buffer size.*

- #define **ENET\_PHY\_MAXADDRESS** (ENET\_MMFR\_PA\_MASK >> ENET\_MMFR\_PA\_SHIFT)  
*Maximum PHY address.*
- #define **ENET\_TX\_INTERRUPT** ((uint32\_t)kENET\_TxFrameInterrupt | (uint32\_t)kENET\_Tx-BufferInterrupt)  
*Enet Tx interrupt flag.*
- #define **ENET\_RX\_INTERRUPT** ((uint32\_t)kENET\_RxFrameInterrupt | (uint32\_t)kENET\_Rx-BufferInterrupt)  
*Enet Rx interrupt flag.*
- #define **ENET\_TS\_INTERRUPT** ((uint32\_t)kENET\_TsTimerInterrupt | (uint32\_t)kENET\_Ts-AvailInterrupt)  
*Enet timestamp interrupt flag.*
- #define **ENET\_ERR\_INTERRUPT**  
*Enet error interrupt flag.*

## Initialization and De-initialization

- void **ENET\_GetDefaultConfig** (enet\_config\_t \*config)  
*Gets the ENET default configuration structure.*
- status\_t **ENET\_Up** (ENET\_Type \*base, enet\_handle\_t \*handle, const enet\_config\_t \*config, const enet\_buffer\_config\_t \*bufferConfig, uint8\_t \*macAddr, uint32\_t srcClock\_Hz)  
*Initializes the ENET module.*
- status\_t **ENET\_Init** (ENET\_Type \*base, enet\_handle\_t \*handle, const enet\_config\_t \*config, const enet\_buffer\_config\_t \*bufferConfig, uint8\_t \*macAddr, uint32\_t srcClock\_Hz)  
*Initializes the ENET module.*
- void **ENET\_Down** (ENET\_Type \*base)  
*Stops the ENET module.*
- void **ENET\_Deinit** (ENET\_Type \*base)  
*Deinitializes the ENET module.*
- static void **ENET\_Reset** (ENET\_Type \*base)  
*Resets the ENET module.*

## MII interface operation

- void **ENET\_SetMII** (ENET\_Type \*base, enet\_mii\_speed\_t speed, enet\_mii\_duplex\_t duplex)  
*Sets the ENET MII speed and duplex.*
- void **ENET\_SetSMI** (ENET\_Type \*base, uint32\_t srcClock\_Hz, bool isPreambleDisabled)  
*Sets the ENET SMI(serial management interface)- MII management interface.*
- static bool **ENET\_GetSMI** (ENET\_Type \*base)  
*Gets the ENET SMI- MII management interface configuration.*
- static uint32\_t **ENET\_ReadSMIData** (ENET\_Type \*base)  
*Reads data from the PHY register through an SMI interface.*
- static void **ENET\_StartSMIWrite** (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, enet\_mii-\_write\_t operation, uint16\_t data)  
*Sends the MDIO IEEE802.3 Clause 22 format write command.*
- static void **ENET\_StartSMIRead** (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, enet\_mii-\_read\_t operation)  
*Sends the MDIO IEEE802.3 Clause 22 format read command.*
- status\_t **ENET\_MDIOWrite** (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t data)  
*MDIO write with IEEE802.3 Clause 22 format.*



- `status_t ENET_MDIORead` (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t \*pData)  
*MDIO read with IEEE802.3 Clause 22 format.*

## MAC Address Filter

- void `ENET_SetMacAddr` (ENET\_Type \*base, uint8\_t \*macAddr)  
*Sets the ENET module Mac address.*
- void `ENET_GetMacAddr` (ENET\_Type \*base, uint8\_t \*macAddr)  
*Gets the ENET module Mac address.*
- void `ENET_AddMulticastGroup` (ENET\_Type \*base, uint8\_t \*address)  
*Adds the ENET device to a multicast group.*
- void `ENET_LeaveMulticastGroup` (ENET\_Type \*base, uint8\_t \*address)  
*Moves the ENET device from a multicast group.*

## Other basic operation

- static void `ENET_ActiveRead` (ENET\_Type \*base)  
*Activates frame reception for multiple rings.*
- static void `ENET_EnableSleepMode` (ENET\_Type \*base, bool enable)  
*Enables/disables the MAC to enter sleep mode.*
- static void `ENET_GetAccelFunction` (ENET\_Type \*base, uint32\_t \*txAccelOption, uint32\_t \*rxAccelOption)  
*Gets ENET transmit and receive accelerator functions from MAC controller.*

## Interrupts.

- static void `ENET_EnableInterrupts` (ENET\_Type \*base, uint32\_t mask)  
*Enables the ENET interrupt.*
- static void `ENET_DisableInterrupts` (ENET\_Type \*base, uint32\_t mask)  
*Disables the ENET interrupt.*
- static uint32\_t `ENET_GetInterruptStatus` (ENET\_Type \*base)  
*Gets the ENET interrupt status flag.*
- static void `ENET_ClearInterruptStatus` (ENET\_Type \*base, uint32\_t mask)  
*Clears the ENET interrupt events status flag.*
- void `ENET_SetRxISRHandler` (ENET\_Type \*base, `enet_isr_t` ISRHandler)  
*Set the second level Rx IRQ handler.*
- void `ENET_SetTxISRHandler` (ENET\_Type \*base, `enet_isr_t` ISRHandler)  
*Set the second level Tx IRQ handler.*
- void `ENET_SetErrISRHandler` (ENET\_Type \*base, `enet_isr_t` ISRHandler)  
*Set the second level Err IRQ handler.*

## Transactional operation

- void `ENET_GetRxErrBeforeReadFrame` (enet\_handle\_t \*handle, `enet_data_error_stats_t` \*eError-Static, uint8\_t ringId)  
*Gets the error statistics of a received frame for ENET specified ring.*
- void `ENET_GetStatistics` (ENET\_Type \*base, `enet_transfer_stats_t` \*statistics)  
*Gets statistical data in transfer.*
- `status_t ENET_GetRxFrameSize` (enet\_handle\_t \*handle, uint32\_t \*length, uint8\_t ringId)

- *Gets the size of the read frame for specified ring.*  
 • [status\\_t ENET\\_ReadFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t \*data, uint32\_t length, uint8\_t ringId, uint32\_t \*ts)  
*Reads a frame from the ENET device.*
- [status\\_t ENET\\_SendFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, const uint8\_t \*data, uint32\_t length, uint8\_t ringId, bool tsFlag, void \*context)  
*Transmits an ENET frame for specified ring.*
- [status\\_t ENET\\_SetTxReclaim](#) (enet\_handle\_t \*handle, bool isEnabled, uint8\_t ringId)  
*Enable or disable tx descriptors reclaim mechanism.*
- void [ENET\\_ReclaimTxDescriptor](#) (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t ringId)  
*Reclaim tx descriptors.*
- [status\\_t ENET\\_GetRxFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, enet\_rx\_frame\_struct\_t \*rxFrame, uint8\_t ringId)  
*Receives one frame in specified BD ring with zero copy.*
- [status\\_t ENET\\_StartTxFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, enet\_tx\_frame\_struct\_t \*txFrame, uint8\_t ringId)  
*Sends one frame in specified BD ring with zero copy.*
- void [ENET\\_TransmitIRQHandler](#) (ENET\_Type \*base, enet\_handle\_t \*handle)  
*The transmit IRQ handler.*
- void [ENET\\_ReceiveIRQHandler](#) (ENET\_Type \*base, enet\_handle\_t \*handle)  
*The receive IRQ handler.*
- void [ENET\\_ErrorIRQHandler](#) (ENET\_Type \*base, enet\_handle\_t \*handle)  
*Some special IRQ handler including the error, mii, wakeup irq handler.*
- void [ENET\\_Ptp1588IRQHandler](#) (ENET\_Type \*base)  
*the common IRQ handler for the 1588 irq handler.*
- void [ENET\\_CommonFrame0IRQHandler](#) (ENET\_Type \*base)  
*the common IRQ handler for the tx/rx/error etc irq handler.*

## 8.4 Data Structure Documentation

### 8.4.1 struct enet\_rx\_bd\_struct\_t

#### Data Fields

- uint16\_t [length](#)  
*Buffer descriptor data length.*
- uint16\_t [control](#)  
*Buffer descriptor control and status.*
- uint32\_t [buffer](#)  
*Data buffer pointer.*



**Field Documentation**

- (1) `uint16_t enet_rx_bd_struct_t::length`
- (2) `uint16_t enet_rx_bd_struct_t::control`
- (3) `uint32_t enet_rx_bd_struct_t::buffer`

**8.4.2 struct enet\_tx\_bd\_struct\_t****Data Fields**

- `uint16_t length`  
*Buffer descriptor data length.*
- `uint16_t control`  
*Buffer descriptor control and status.*
- `uint32_t buffer`  
*Data buffer pointer.*

**Field Documentation**

- (1) `uint16_t enet_tx_bd_struct_t::length`
- (2) `uint16_t enet_tx_bd_struct_t::control`
- (3) `uint32_t enet_tx_bd_struct_t::buffer`

**8.4.3 struct enet\_data\_error\_stats\_t****Data Fields**

- `uint32_t statsRxLenGreaterErr`  
*Receive length greater than RCR[MAX\_FL].*
- `uint32_t statsRxAlignErr`  
*Receive non-octet alignment/.*
- `uint32_t statsRxFcsErr`  
*Receive CRC error.*
- `uint32_t statsRxOverRunErr`  
*Receive over run.*
- `uint32_t statsRxTruncateErr`  
*Receive truncate.*

**Field Documentation**

- (1) `uint32_t enet_data_error_stats_t::statsRxLenGreaterErr`
- (2) `uint32_t enet_data_error_stats_t::statsRxFcsErr`
- (3) `uint32_t enet_data_error_stats_t::statsRxOverRunErr`
- (4) `uint32_t enet_data_error_stats_t::statsRxTruncateErr`

**8.4.4 struct enet\_rx\_frame\_error\_t****Data Fields**

- `bool statsRxTruncateErr`: 1  
*Receive truncate.*
- `bool statsRxOverRunErr`: 1  
*Receive over run.*
- `bool statsRxFcsErr`: 1  
*Receive CRC error.*
- `bool statsRxAlignErr`: 1  
*Receive non-octet alignment.*
- `bool statsRxLenGreaterErr`: 1  
*Receive length greater than RCR[MAX\_FL].*

**Field Documentation**

- (1) `bool enet_rx_frame_error_t::statsRxTruncateErr`
- (2) `bool enet_rx_frame_error_t::statsRxOverRunErr`
- (3) `bool enet_rx_frame_error_t::statsRxFcsErr`
- (4) `bool enet_rx_frame_error_t::statsRxAlignErr`
- (5) `bool enet_rx_frame_error_t::statsRxLenGreaterErr`

**8.4.5 struct enet\_transfer\_stats\_t****Data Fields**

- `uint32_t statsRxFrameCount`  
*Rx frame number.*
- `uint32_t statsRxFrameOk`  
*Good Rx frame number.*
- `uint32_t statsRxCrcErr`  
*Rx frame number with CRC error.*
- `uint32_t statsRxAlignErr`  
*Rx frame number with alignment error.*

- uint32\_t [statsRxDropInvalidSFD](#)  
*Dropped frame number due to invalid SFD.*
- uint32\_t [statsRxFifoOverflowErr](#)  
*Rx FIFO overflow count.*
- uint32\_t [statsTxFrameCount](#)  
*Tx frame number.*
- uint32\_t [statsTxFrameOk](#)  
*Good Tx frame number.*
- uint32\_t [statsTx\\_crc\\_align\\_err](#)  
*The transmit frame is error.*
- uint32\_t [statsTxFifoUnderRunErr](#)  
*Tx FIFO underrun count.*

### Field Documentation

- (1) uint32\_t enet\_transfer\_stats\_t::statsRxFrameCount
- (2) uint32\_t enet\_transfer\_stats\_t::statsRxFrameOk
- (3) uint32\_t enet\_transfer\_stats\_t::statsRx\_crc\_err
- (4) uint32\_t enet\_transfer\_stats\_t::statsRx\_align\_err
- (5) uint32\_t enet\_transfer\_stats\_t::statsRxDropInvalidSFD
- (6) uint32\_t enet\_transfer\_stats\_t::statsRxFifoOverflowErr
- (7) uint32\_t enet\_transfer\_stats\_t::statsTxFrameCount
- (8) uint32\_t enet\_transfer\_stats\_t::statsTxFrameOk
- (9) uint32\_t enet\_transfer\_stats\_t::statsTx\_crc\_align\_err
- (10) uint32\_t enet\_transfer\_stats\_t::statsTxFifoUnderRunErr

## 8.4.6 struct enet\_frame\_info\_t

### Data Fields

- void \* [context](#)  
*User specified data.*

## 8.4.7 struct enet\_tx\_dirty\_ring\_t

### Data Fields

- [enet\\_frame\\_info\\_t](#) \* [txDirtyBase](#)  
*Dirty buffer descriptor base address pointer.*

- uint16\_t [txGenIdx](#)  
*tx generate index.*
- uint16\_t [txConsumIdx](#)  
*tx consume index.*
- uint16\_t [txRingLen](#)  
*tx ring length.*
- bool [isFull](#)  
*tx ring is full flag.*

### Field Documentation

- (1) `enet_frame_info_t* enet_tx_dirty_ring_t::txDirtyBase`
- (2) `uint16_t enet_tx_dirty_ring_t::txGenIdx`
- (3) `uint16_t enet_tx_dirty_ring_t::txConsumIdx`
- (4) `uint16_t enet_tx_dirty_ring_t::txRingLen`
- (5) `bool enet_tx_dirty_ring_t::isFull`

### 8.4.8 struct enet\_buffer\_config\_t

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

1. The aligned receive and transmit buffer size must be evenly divisible by ENET\_BUFF\_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.
2. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET\_BUFF\_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
3. The aligned transmit and receive data buffer start address must be evenly divisible by ENET\_BUFF\_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber \* rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber \* txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.

### Data Fields

- uint16\_t [rxBdNumber](#)  
*Receive buffer descriptor number.*
- uint16\_t [txBdNumber](#)  
*Transmit buffer descriptor number.*
- uint16\_t [rxBuffSizeAlign](#)  
*Aligned receive data buffer size.*
- uint16\_t [txBuffSizeAlign](#)  
*Aligned transmit data buffer size.*

- volatile `enet_rx_bd_struct_t * rxBdStartAddrAlign`  
*Aligned receive buffer descriptor start address: should be non-cacheable.*
- volatile `enet_tx_bd_struct_t * txBdStartAddrAlign`  
*Aligned transmit buffer descriptor start address: should be non-cacheable.*
- `uint8_t * rxBufferAlign`  
*Receive data buffer start address.*
- `uint8_t * txBufferAlign`  
*Transmit data buffer start address.*
- `bool rxMaintainEnable`  
*Receive buffer cache maintain.*
- `bool txMaintainEnable`  
*Transmit buffer cache maintain.*
- `enet_frame_info_t * txFrameInfo`  
*Transmit frame information start address.*

### Field Documentation

- (1) `uint16_t enet_buffer_config_t::rxBdNumber`
- (2) `uint16_t enet_buffer_config_t::txBdNumber`
- (3) `uint16_t enet_buffer_config_t::rxBuffSizeAlign`
- (4) `uint16_t enet_buffer_config_t::txBuffSizeAlign`
- (5) `volatile enet_rx_bd_struct_t* enet_buffer_config_t::rxBdStartAddrAlign`
- (6) `volatile enet_tx_bd_struct_t* enet_buffer_config_t::txBdStartAddrAlign`
- (7) `uint8_t* enet_buffer_config_t::rxBufferAlign`
- (8) `uint8_t* enet_buffer_config_t::txBufferAlign`
- (9) `bool enet_buffer_config_t::rxMaintainEnable`
- (10) `bool enet_buffer_config_t::txMaintainEnable`
- (11) `enet_frame_info_t* enet_buffer_config_t::txFrameInfo`

### 8.4.9 struct enet\_config\_t

Note:

1. `macSpecialConfig` is used for a special control configuration, A logical OR of "`enet_special_control_flag_t`". For a special configuration for MAC, set this parameter to 0.
2. `txWatermark` is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of `txWatermark` is 0x2F - 4032 bytes written to TX FIFO .... `txWatermark` allows minimizing the transmit latency to set the `txWatermark` to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.

3. `rxFifoFullThreshold` is similar to the `txWatermark` for cut-through operation in RX. It is in 64-bit words. The minimum is `ENET_FIFO_MIN_RX_FULL` and the maximum is `0xFF`. If the end of the frame is stored in FIFO and the frame size is smaller than the `txWatermark`, the frame is still transmitted. The rule is the same for `rxFifoFullThreshold` in the receive direction.
4. When "`kENET_ControlFlowControlEnable`" is set in the `macSpecialConfig`, ensure that the `pauseDuration`, `rxFifoEmptyThreshold`, and `rxFifoStatEmptyThreshold` are set for flow control enabled case.
5. When "`kENET_ControlStoreAndFwdDisabled`" is set in the `macSpecialConfig`, ensure that the `rxFifoFullThreshold` and `txFifoWatermark` are set for store and forward disable.
6. The `rxAccelerConfig` and `txAccelerConfig` default setting with 0 - accelerator are disabled. The "`enet_tx_accelerator_t`" and "`enet_rx_accelerator_t`" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, `kENET_ControlStoreAndFwdDisabled` should not be set.
7. The `intCoalesceCfg` can be used in the rx or tx enabled cases to decrease the CPU loading.

## Data Fields

- `uint32_t macSpecialConfig`  
*Mac special configuration.*
- `uint32_t interrupt`  
*Mac interrupt source.*
- `uint16_t rxMaxFrameLen`  
*Receive maximum frame length.*
- `enet_mii_mode_t miiMode`  
*MII mode.*
- `enet_mii_speed_t miiSpeed`  
*MII Speed.*
- `enet_mii_duplex_t miiDuplex`  
*MII duplex.*
- `uint8_t rxAccelerConfig`  
*Receive accelerator, A logical OR of "enet\_rx\_accelerator\_t".*
- `uint8_t txAccelerConfig`  
*Transmit accelerator, A logical OR of "enet\_rx\_accelerator\_t".*
- `uint16_t pauseDuration`  
*For flow control enabled case: Pause duration.*
- `uint8_t rxFifoEmptyThreshold`  
*For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.*
- `uint8_t rxFifoFullThreshold`  
*For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.*
- `uint8_t txFifoWatermark`  
*For store and forward disable case, the data required in TX FIFO before a frame transmit start.*
- `uint8_t ringNum`  
*Number of used rings.*
- `enet_rx_alloc_callback_t rxBuffAlloc`  
*Callback function to alloc memory, must be provided for zero-copy Rx.*

- [enet\\_rx\\_free\\_callback\\_t rxBuffFree](#)  
*Callback function to free memory, must be provided for zero-copy Rx.*
- [enet\\_callback\\_t callback](#)  
*General callback function.*
- void \* [userData](#)  
*Callback function parameter.*

## Field Documentation

### (1) `uint32_t enet_config_t::macSpecialConfig`

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

### (2) `uint32_t enet_config_t::interrupt`

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

### (3) `uint16_t enet_config_t::rxMaxFrameLen`

### (4) `enet_mii_mode_t enet_config_t::miiMode`

### (5) `enet_mii_speed_t enet_config_t::miiSpeed`

### (6) `enet_mii_duplex_t enet_config_t::miiDuplex`

### (7) `uint8_t enet_config_t::rxAccelerConfig`

### (8) `uint8_t enet_config_t::txAccelerConfig`

### (9) `uint16_t enet_config_t::pauseDuration`

### (10) `uint8_t enet_config_t::rxFifoEmptyThreshold`

### (11) `uint8_t enet_config_t::rxFifoFullThreshold`

### (12) `uint8_t enet_config_t::txFifoWatermark`

### (13) `uint8_t enet_config_t::ringNum`

default with 1 – single ring.

(14) `enet_rx_alloc_callback_t enet_config_t::rxBuffAlloc`

(15) `enet_rx_free_callback_t enet_config_t::rxBuffFree`

(16) `enet_callback_t enet_config_t::callback`

(17) `void* enet_config_t::userData`

#### 8.4.10 struct `enet_tx_bd_ring_t`

##### Data Fields

- volatile `enet_tx_bd_struct_t * txBdBase`  
*Buffer descriptor base address pointer.*
- `uint16_t txGenIdx`  
*The current available transmit buffer descriptor pointer.*
- `uint16_t txConsumeIdx`  
*Transmit consume index.*
- volatile `uint16_t txDescUsed`  
*Transmit descriptor used number.*
- `uint16_t txRingLen`  
*Transmit ring length.*

##### Field Documentation

(1) `volatile enet_tx_bd_struct_t* enet_tx_bd_ring_t::txBdBase`

(2) `uint16_t enet_tx_bd_ring_t::txGenIdx`

(3) `uint16_t enet_tx_bd_ring_t::txConsumeIdx`

(4) `volatile uint16_t enet_tx_bd_ring_t::txDescUsed`

(5) `uint16_t enet_tx_bd_ring_t::txRingLen`

#### 8.4.11 struct `enet_rx_bd_ring_t`

##### Data Fields

- volatile `enet_rx_bd_struct_t * rxBdBase`  
*Buffer descriptor base address pointer.*
- `uint16_t rxGenIdx`  
*The current available receive buffer descriptor pointer.*
- `uint16_t rxRingLen`  
*Receive ring length.*



## Field Documentation

- (1) `volatile enet_rx_bd_struct_t* enet_rx_bd_ring_t::rxBdBase`
- (2) `uint16_t enet_rx_bd_ring_t::rxGenIdx`
- (3) `uint16_t enet_rx_bd_ring_t::rxRingLen`

8.4.12 `struct _enet_handle`

## Data Fields

- `enet_rx_bd_ring_t rxBdRing` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer descriptor.*
- `enet_tx_bd_ring_t txBdRing` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer descriptor.*
- `uint16_t rxBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer size alignment.*
- `uint16_t txBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer size alignment.*
- `bool rxMaintainEnable` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer cache maintain.*
- `bool txMaintainEnable` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer cache maintain.*
- `uint8_t ringNum`  
*Number of used rings.*
- `enet_callback_t callback`  
*Callback function.*
- `void * userData`  
*Callback function parameter.*
- `enet_tx_dirty_ring_t txDirtyRing` [FSL\_FEATURE\_ENET\_QUEUE]  
*Ring to store tx frame information.*
- `bool txReclaimEnable` [FSL\_FEATURE\_ENET\_QUEUE]  
*Tx reclaim enable flag.*
- `enet_rx_alloc_callback_t rxBuffAlloc`  
*Callback function to alloc memory for zero copy Rx.*
- `enet_rx_free_callback_t rxBuffFree`  
*Callback function to free memory for zero copy Rx.*
- `uint8_t multicastCount` [64]  
*Multicast collisions counter.*

**Field Documentation**

- (1) **enet\_rx\_bd\_ring\_t enet\_handle\_t::rxBdRing[FSL\_FEATURE\_ENET\_QUEUE]**
- (2) **enet\_tx\_bd\_ring\_t enet\_handle\_t::txBdRing[FSL\_FEATURE\_ENET\_QUEUE]**
- (3) **uint16\_t enet\_handle\_t::rxBuffSizeAlign[FSL\_FEATURE\_ENET\_QUEUE]**
- (4) **uint16\_t enet\_handle\_t::txBuffSizeAlign[FSL\_FEATURE\_ENET\_QUEUE]**
- (5) **bool enet\_handle\_t::rxMaintainEnable[FSL\_FEATURE\_ENET\_QUEUE]**
- (6) **bool enet\_handle\_t::txMaintainEnable[FSL\_FEATURE\_ENET\_QUEUE]**
- (7) **uint8\_t enet\_handle\_t::ringNum**
- (8) **enet\_callback\_t enet\_handle\_t::callback**
- (9) **void\* enet\_handle\_t::userData**
- (10) **enet\_tx\_dirty\_ring\_t enet\_handle\_t::txDirtyRing[FSL\_FEATURE\_ENET\_QUEUE]**
- (11) **bool enet\_handle\_t::txReclaimEnable[FSL\_FEATURE\_ENET\_QUEUE]**
- (12) **enet\_rx\_alloc\_callback\_t enet\_handle\_t::rxBuffAlloc**
- (13) **enet\_rx\_free\_callback\_t enet\_handle\_t::rxBuffFree**

## 8.5 Macro Definition Documentation

8.5.1 **#define FSL\_ENET\_DRIVER\_VERSION (MAKE\_VERSION(2, 7, 0))**

8.5.2 **#define FSL\_FEATURE\_ENET\_QUEUE 1 /\* Singal queue for previous IP. \*/**

8.5.3 **#define ENET\_BUFFDESCRIPTOR\_RX\_EMPTY\_MASK 0x8000U**

8.5.4 **#define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER1\_MASK 0x4000U**

8.5.5 **#define ENET\_BUFFDESCRIPTOR\_RX\_WRAP\_MASK 0x2000U**

8.5.6 **#define ENET\_BUFFDESCRIPTOR\_RX\_SOFTOWNER2\_Mask 0x1000U**

8.5.7 **#define ENET\_BUFFDESCRIPTOR\_RX\_LAST\_MASK 0x0800U**

8.5.8 **#define ENET\_BUFFDESCRIPTOR\_RX\_MISS\_MASK 0x0100U**

8.5.9 **#define ENET\_BUFFDESCRIPTOR\_RX\_BROADCAST\_MASK 0x0080U**

8.5.10 **#define ENET\_BUFFDESCRIPTOR\_RX\_MULTICAST\_MASK 0x0040U**

8.5.11 **#define ENET\_BUFFDESCRIPTOR\_RX\_LENVIOLATE\_MASK 0x0020U**

8.5.12 **#define ENET\_BUFFDESCRIPTOR\_RX\_NOOCTET\_MASK 0x0010U**

8.5.13 **#define ENET\_BUFFDESCRIPTOR\_RX\_CRC\_MASK 0x0004U**

8.5.14 **#define ENET\_BUFFDESCRIPTOR\_RX\_OVERRUN\_MASK 0x0002U**

8.5.15 **#define ENET\_BUFFDESCRIPTOR\_RX\_TRUNC\_MASK 0x0001U**

8.5.16 **#define ENET\_BUFFDESCRIPTOR\_TX\_READY\_MASK 0x8000U**

8.5.17 **#define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWENER1\_MASK 0x4000U**

8.5.18 **#define ENET\_BUFFDESCRIPTOR\_TX\_WRAP\_MASK 0x2000U**

8.5.19 **#define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWENER2\_MASK 0x1000U**

8.5.20 **#define ENET\_BUFFDESCRIPTOR\_TX\_LAST\_MASK 0x0800U**

8.5.21 **#define ENET\_BUFFDESCRIPTOR\_TX\_TRANSMITCRC\_MASK 0x0400U**

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

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

**8.5.24 #define ENET\_FRAME\_VLAN\_TAGLEN 4U**

**8.5.25 #define ENET\_FRAME\_CRC\_LEN 4U**

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

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

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

**8.5.29 #define ENET\_TX\_INTERRUPT ((uint32\_t)kENET\_TxFrameInterrupt |  
(uint32\_t)kENET\_TxBufferInterrupt)**

**8.5.30 #define ENET\_RX\_INTERRUPT ((uint32\_t)kENET\_RxFrameInterrupt |  
(uint32\_t)kENET\_RxBufferInterrupt)**

**8.5.31 #define ENET\_TS\_INTERRUPT ((uint32\_t)kENET\_TsTimerInterrupt |  
(uint32\_t)kENET\_TsAvailInterrupt)**

**8.5.32 #define ENET\_ERR\_INTERRUPT**

**Value:**

```
((uint32_t)kENET_BabrInterrupt | (uint32_t)
 kENET_BabtInterrupt | (uint32_t)kENET_EBusERInterrupt | \
 (uint32_t)kENET_LateCollisionInterrupt | (uint32_t)
 kENET_RetryLimitInterrupt | \
 (uint32_t)kENET_UnderrunInterrupt | (uint32_t)
 kENET_PayloadRxInterrupt)
```

## 8.6 Typedef Documentation

**8.6.1** `typedef void>(* enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)`

**8.6.2** `typedef void(* enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)`

**8.6.3** `typedef void(* enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)`

**8.6.4** `typedef void(* enet_isr_t)(ENET_Type *base, enet_handle_t *handle)`

## 8.7 Enumeration Type Documentation

### 8.7.1 anonymous enum

Enumerator

***kStatus\_ENET\_InitMemoryFail*** Init fails since buffer memory is not enough.

***kStatus\_ENET\_RxFrameError*** A frame received but data error happen.

***kStatus\_ENET\_RxFrameFail*** Failed to receive a frame.

***kStatus\_ENET\_RxFrameEmpty*** No frame arrive.

***kStatus\_ENET\_RxFrameDrop*** Rx frame is dropped since no buffer memory.

***kStatus\_ENET\_TxFrameOverLen*** Tx frame over length.

***kStatus\_ENET\_TxFrameBusy*** Tx buffer descriptors are under process.

***kStatus\_ENET\_TxFrameFail*** Transmit frame fail.

### 8.7.2 enum enet\_mii\_mode\_t

Enumerator

***kENET\_MiiMode*** MII mode for data interface.

***kENET\_RmiiMode*** RMII mode for data interface.

### 8.7.3 enum enet\_mii\_speed\_t

Notice: "kENET\_MiiSpeed1000M" only supported when mii mode is "kENET\_RgmiiMode".

Enumerator

***kENET\_MiiSpeed10M*** Speed 10 Mbps.

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

### 8.7.4 enum enet\_mii\_duplex\_t

Enumerator

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

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

### 8.7.5 enum enet\_mii\_write\_t

Enumerator

***kENET\_MiiWriteNoCompliant*** Write frame operation, but not MII-compliant.

***kENET\_MiiWriteValidFrame*** Write frame operation for a valid MII management frame.

### 8.7.6 enum enet\_mii\_read\_t

Enumerator

***kENET\_MiiReadValidFrame*** Read frame operation for a valid MII management frame.

***kENET\_MiiReadNoCompliant*** Read frame operation, but not MII-compliant.

### 8.7.7 enum enet\_special\_control\_flag\_t

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the [enet\\_config\\_t](#). The `kENET_ControlStoreAndFwdDisable` is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure `rxFifoFullThreshold` and `txFifoWatermark` in the [enet\\_config\\_t](#).

Enumerator

***kENET\_ControlFlowControlEnable*** Enable ENET flow control: pause frame.

***kENET\_ControlRxPayloadCheckEnable*** Enable ENET receive payload length check.

***kENET\_ControlRxPadRemoveEnable*** Padding is removed from received frames.

***kENET\_ControlRxBroadcastRejectEnable*** Enable broadcast frame reject.

***kENET\_ControlMacAddrInsert*** Enable MAC address insert.

***kENET\_ControlStoreAndFwdDisable*** Enable FIFO store and forward.

***kENET\_ControlSMIPreambleDisable*** Enable SMI preamble.

***kENET\_ControlPromiscuousEnable*** Enable promiscuous mode.

***kENET\_ControlMIILoopEnable*** Enable ENET MII loop back.

***kENET\_ControlVLANTagEnable*** Enable normal VLAN (single vlan tag).

### 8.7.8 enum enet\_interrupt\_enable\_t

This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Enumerator

***kENET\_BabrInterrupt*** Babbling receive error interrupt source.  
***kENET\_BabtInterrupt*** Babbling transmit error interrupt source.  
***kENET\_GraceStopInterrupt*** Graceful stop complete interrupt source.  
***kENET\_TxFrameInterrupt*** TX FRAME interrupt source.  
***kENET\_TxBufferInterrupt*** TX BUFFER interrupt source.  
***kENET\_RxFrameInterrupt*** RX FRAME interrupt source.  
***kENET\_RxBufferInterrupt*** RX BUFFER interrupt source.  
***kENET\_MiiInterrupt*** MII interrupt source.  
***kENET\_EBusERInterrupt*** Ethernet bus error interrupt source.  
***kENET\_LateCollisionInterrupt*** Late collision interrupt source.  
***kENET\_RetryLimitInterrupt*** Collision Retry Limit interrupt source.  
***kENET\_UnderrunInterrupt*** Transmit FIFO underrun interrupt source.  
***kENET\_PayloadRxInterrupt*** Payload Receive error interrupt source.  
***kENET\_WakeupInterrupt*** WAKEUP interrupt source.  
***kENET\_TsAvailInterrupt*** TS AVAIL interrupt source for PTP.  
***kENET\_TsTimerInterrupt*** TS WRAP interrupt source for PTP.

### 8.7.9 enum enet\_event\_t

Enumerator

***kENET\_RxEvent*** Receive event.  
***kENET\_TxEvent*** Transmit event.  
***kENET\_ErrEvent*** Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .  
***kENET\_WakeUpEvent*** Wake up from sleep mode event.  
***kENET\_TimeStampEvent*** Time stamp event.  
***kENET\_TimeStampAvailEvent*** Time stamp available event.

### 8.7.10 enum enet\_tx\_accelerator\_t

Enumerator

***kENET\_TxAccellShift16Enabled*** Transmit FIFO shift-16.  
***kENET\_TxAccellIpCheckEnabled*** Insert IP header checksum.  
***kENET\_TxAccelProtoCheckEnabled*** Insert protocol checksum.

### 8.7.11 enum enet\_rx\_accelerator\_t

Enumerator

*kENET\_RxAccelPadRemoveEnabled* Padding removal for short IP frames.  
*kENET\_RxAccelIpCheckEnabled* Discard with wrong IP header checksum.  
*kENET\_RxAccelProtoCheckEnabled* Discard with wrong protocol checksum.  
*kENET\_RxAccelMacCheckEnabled* Discard with Mac layer errors.  
*kENET\_RxAccelIsShift16Enabled* Receive FIFO shift-16.

## 8.8 Function Documentation

### 8.8.1 uint32\_t ENET\_GetInstance ( ENET\_Type \* *base* )

Parameters

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

Returns

ENET instance.

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

The purpose of this API is to get the default ENET MAC controller configure structure for [ENET\\_Init\(\)](#). User may use the initialized structure unchanged in [ENET\\_Init\(\)](#), or modify some fields of the structure before calling [ENET\\_Init\(\)](#). Example:

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

Parameters

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

### 8.8.3 status\_t ENET\_Up ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, const enet\_config\_t \* *config*, const enet\_buffer\_config\_t \* *bufferConfig*, uint8\_t \* *macAddr*, uint32\_t *srcClock\_Hz* )

This function initializes the module with the ENET configuration.



## Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" and calling ENET\_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET\\_Up\(\)](#).

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ENET peripheral base address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <i>handle</i>       | ENET handler pointer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>config</i>       | ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.                                                                                                                                                                                                                                                                                      |
| <i>bufferConfig</i> | ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer. |
| <i>macAddr</i>      | ENET mac address of Ethernet device. This MAC address should be provided.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>srcClock_Hz</i>  | The internal module clock source for MII clock.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to initialize the ethernet driver.    |
| <i>kStatus_ENET_Init-MemoryFail</i> | Init fails since buffer memory is not enough. |

**8.8.4** `status_t ENET_Init ( ENET_Type * base, enet_handle_t * handle, const enet_config_t * config, const enet_buffer_config_t * bufferConfig, uint8_t * macAddr, uint32_t srcClock_Hz )`

This function ungates the module clock and initializes it with the ENET configuration.

## Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" and calling ENET\_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET\\_Init\(\)](#).

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ENET peripheral base address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <i>handle</i>       | ENET handler pointer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>config</i>       | ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.                                                                                                                                                                                                                                                                                      |
| <i>bufferConfig</i> | ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer. |
| <i>macAddr</i>      | ENET mac address of Ethernet device. This MAC address should be provided.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>srcClock_Hz</i>  | The internal module clock source for MII clock.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to initialize the ethernet driver.    |
| <i>kStatus_ENET_Init-MemoryFail</i> | Init fails since buffer memory is not enough. |

**8.8.5 void ENET\_Down ( ENET\_Type \* *base* )**

This function disables the ENET module.

## Parameters

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

**8.8.6 void ENET\_Deinit ( ENET\_Type \* *base* )**

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

## Parameters

---

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

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

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

Parameters

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

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

This API is provided to dynamically change the speed and duplex for MAC.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>speed</i>  | The speed of the RMII mode.   |
| <i>duplex</i> | The duplex of the RMII mode.  |

### 8.8.9 void ENET\_SetSMI ( ENET\_Type \* *base*, uint32\_t *srcClock\_Hz*, bool *isPreambleDisabled* )

Parameters

|                            |                                                                                                                                                   |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>                | ENET peripheral base address.                                                                                                                     |
| <i>srcClock_Hz</i>         | This is the ENET module clock frequency. See clock distribution.                                                                                  |
| <i>isPreamble-Disabled</i> | The preamble disable flag. <ul style="list-style-type: none"> <li>• true Enables the preamble.</li> <li>• false Disables the preamble.</li> </ul> |

### 8.8.10 static bool ENET\_GetSMI ( ENET\_Type \* *base* ) [inline], [static]

This API is used to get the SMI configuration to check whether the MII management interface has been set.

## Parameters

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

## Returns

The SMI setup status true or false.

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

## Parameters

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

## Returns

The data read from PHY

### 8.8.12 static void ENET\_StartSMIWrite ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, enet\_mii\_write\_t *operation*, uint16\_t *data* ) [inline], [static]

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIOWrite\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

## Parameters

|                  |                                              |
|------------------|----------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                |
| <i>phyAddr</i>   | The PHY address. Range from 0 ~ 31.          |
| <i>regAddr</i>   | The PHY register address. Range from 0 ~ 31. |
| <i>operation</i> | The write operation.                         |
| <i>data</i>      | The data written to PHY.                     |

### 8.8.13 static void ENET\_StartSMIRead ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, enet\_mii\_read\_t *operation* ) [inline], [static]

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIORead\(\)](#) can be called. For customized requirements,

implement with combining separated APIs.

## Parameters

|                  |                                              |
|------------------|----------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                |
| <i>phyAddr</i>   | The PHY address. Range from 0 ~ 31.          |
| <i>regAddr</i>   | The PHY register address. Range from 0 ~ 31. |
| <i>operation</i> | The read operation.                          |

#### 8.8.14 status\_t ENET\_MDIOWrite ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, uint16\_t *data* )

## Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | ENET peripheral base address.        |
| <i>phyAddr</i> | The PHY address. Range from 0 ~ 31.  |
| <i>regAddr</i> | The PHY register. Range from 0 ~ 31. |
| <i>data</i>    | The data written to PHY.             |

## Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

#### 8.8.15 status\_t ENET\_MDIORead ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, uint16\_t \* *pData* )

## Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | ENET peripheral base address.        |
| <i>phyAddr</i> | The PHY address. Range from 0 ~ 31.  |
| <i>regAddr</i> | The PHY register. Range from 0 ~ 31. |
| <i>pData</i>   | The data read from PHY.              |

## Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

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



## Parameters

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

**8.8.17 void ENET\_GetMacAddr ( ENET\_Type \* *base*, uint8\_t \* *macAddr* )**

## Parameters

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

**8.8.18 void ENET\_AddMulticastGroup ( ENET\_Type \* *base*, uint8\_t \* *address* )**

## Parameters

|                |                                                                        |
|----------------|------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                          |
| <i>address</i> | The six-byte multicast group address which is provided by application. |

**8.8.19 void ENET\_LeaveMulticastGroup ( ENET\_Type \* *base*, uint8\_t \* *address* )**

## Parameters

|                |                                                                        |
|----------------|------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                          |
| <i>address</i> | The six-byte multicast group address which is provided by application. |

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

This function is to active the enet read process.

## Note

This must be called after the MAC configuration and state are ready. It must be called after the [ENET\\_Init\(\)](#). This should be called when the frame reception is required.

## Parameters

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

### 8.8.21 static void ENET\_EnableSleepMode ( ENET\_Type \* *base*, bool *enable* ) [inline], [static]

This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

## Parameters

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

### 8.8.22 static void ENET\_GetAccelFunction ( ENET\_Type \* *base*, uint32\_t \* *txAccelOption*, uint32\_t \* *rxAccelOption* ) [inline], [static]

## Parameters

|                      |                                                                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | ENET peripheral base address.                                                                                                                  |
| <i>txAccelOption</i> | The transmit accelerator option. The "enet_tx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option. |
| <i>rxAccelOption</i> | The receive accelerator option. The "enet_rx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.  |

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

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See [enet\\_interrupt\\_enable\\_t](#). For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
* ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt |
* kENET_RxFrameInterrupt);
```

## Parameters

|             |                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                |
| <i>mask</i> | ENET interrupts to enable. This is a logical OR of the enumeration <a href="#">enet_interrupt_enable_t</a> . |

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

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [enet\\_interrupt\\_enable\\_t](#). For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
* ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt |
* kENET_RxFrameInterrupt);
*
```

## Parameters

|             |                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                 |
| <i>mask</i> | ENET interrupts to disable. This is a logical OR of the enumeration <a href="#">enet_interrupt_enable_t</a> . |

#### 8.8.25 static uint32\_t ENET\_GetInterruptStatus ( ENET\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration [enet\\_interrupt\\_enable\\_t](#).

#### 8.8.26 static void ENET\_ClearInterruptStatus ( ENET\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the [enet\\_interrupt\\_enable\\_t](#). For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```

* ENET_ClearInterruptStatus(ENET,
* kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
*

```

Parameters

|             |                                                                                                                                     |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                                       |
| <i>mask</i> | ENET interrupt source to be cleared. This is the logical OR of members of the enumeration <a href="#">enet_interrupt_enable_t</a> . |

### 8.8.27 void ENET\_SetRxISRHandler ( ENET\_Type \* *base*, enet\_isr\_t *ISRHandler* )

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | ENET peripheral base address. |
| <i>ISRHandler</i> | The handler to install.       |

### 8.8.28 void ENET\_SetTxISRHandler ( ENET\_Type \* *base*, enet\_isr\_t *ISRHandler* )

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | ENET peripheral base address. |
| <i>ISRHandler</i> | The handler to install.       |

### 8.8.29 void ENET\_SetErrISRHandler ( ENET\_Type \* *base*, enet\_isr\_t *ISRHandler* )

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | ENET peripheral base address. |
| <i>ISRHandler</i> | The handler to install.       |

### 8.8.30 void ENET\_GetRxErrBeforeReadFrame ( enet\_handle\_t \* *handle*, enet\_data\_error\_stats\_t \* *eErrorStatic*, uint8\_t *ringId* )

This API must be called after the [ENET\\_GetRxFramSize](#) and before the [ENET\\_ReadFrame\(\)](#). If the [ENET\\_GetRxFramSize](#) returns [kStatus\\_ENET\\_RxFrameError](#), the [ENET\\_GetRxErrBeforeReadFrame](#) can be used to get the exact error statistics. This is an example.

```

* status = ENET_GetRxFrameSize(&g_handle, &length, 0);
* if (status == kStatus_ENET_RxFrameError)
* {
* Comments: Get the error information of the received frame.
* ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
* Comments: update the receive buffer.
* ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
* }
*

```

## Parameters

|                     |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <i>handle</i>       | The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init. |
| <i>eErrorStatic</i> | The error statistics structure pointer.                                                     |
| <i>ringId</i>       | The ring index, range from 0 ~ (FSL_FEATURE_ENET_INSTANCE_QUEUEEn(x) - 1).                  |

### 8.8.31 void ENET\_GetStatistics ( ENET\_Type \* *base*, enet\_transfer\_stats\_t \* *statistics* )

## Parameters

|                   |                                   |
|-------------------|-----------------------------------|
| <i>base</i>       | ENET peripheral base address.     |
| <i>statistics</i> | The statistics structure pointer. |

### 8.8.32 status\_t ENET\_GetRxFrameSize ( enet\_handle\_t \* *handle*, uint32\_t \* *length*, uint8\_t *ringId* )

This function gets a received frame size from the ENET buffer descriptors.

## Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET\_GetRxFrameSize, [ENET\\_ReadFrame\(\)](#) should be called to receive frame and update the BD if the result is not "kStatus\_ENET\_RxFrameEmpty".

## Parameters

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>handle</i> | The ENET handler structure. This is the same handler pointer used in the ENET_Init. |
| <i>length</i> | The length of the valid frame received.                                             |
| <i>ringId</i> | The ring index or ring number.                                                      |

## Return values

|                                   |                                                                                                                                      |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_ENET_RxFrame-Empty</i> | No frame received. Should not call ENET_ReadFrame to read frame.                                                                     |
| <i>kStatus_ENET_RxFrame-Error</i> | Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.                    |
| <i>kStatus_Success</i>            | Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input. |

### 8.8.33 status\_t ENET\_ReadFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint8\_t \* *data*, uint32\_t *length*, uint8\_t *ringId*, uint32\_t \* *ts* )

This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL.

## Note

It doesn't store the timestamp in the receive timestamp queue. The ENET\_GetRxFrameSize should be used to get the size of the prepared data buffer. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption. This is an example:

```
* uint32_t length;
* enet_handle_t g_handle;
* Comments: Get the received frame size firstly.
* status = ENET_GetRxFrameSize(&g_handle, &length, 0);
* if (length != 0)
* {
* Comments: Allocate memory here with the size of "length"
* uint8_t *data = memory allocate interface;
* if (!data)
* {
* ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
* Comments: Add the console warning log.
* }
* else
* {
* status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
* Comments: Call stack input API to deliver the data to stack
* }
* }
* else if (status == kStatus_ENET_RxFrameError)
* {
* Comments: Update the received buffer when a error frame is received.
```

```

* ENET_ReadFrame (ENET, &g_handle, NULL, 0, 0, NULL);
* }
*

```

## Parameters

|               |                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                                      |
| <i>handle</i> | The ENET handler structure. This is the same handler pointer used in the ENET_Init.                |
| <i>data</i>   | The data buffer provided by user to store the frame which memory size should be at least "length". |
| <i>length</i> | The size of the data buffer which is still the length of the received frame.                       |
| <i>ringId</i> | The ring index or ring number.                                                                     |
| <i>ts</i>     | The timestamp address to store received timestamp.                                                 |

## Returns

The execute status, successful or failure.

### 8.8.34 **status\_t ENET\_SendFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, const uint8\_t \* *data*, uint32\_t *length*, uint8\_t *ringId*, bool *tsFlag*, void \* *context* )**

## Note

The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption.

## Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                     |
| <i>handle</i> | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>data</i>   | The data buffer provided by user to send.                                         |
| <i>length</i> | The length of the data to send.                                                   |

|                |                                                         |
|----------------|---------------------------------------------------------|
| <i>ringId</i>  | The ring index or ring number.                          |
| <i>tsFlag</i>  | Timestamp enable flag.                                  |
| <i>context</i> | Used by user to handle some events after transmit over. |

## Return values

|                                  |                                                                                                                                                                                                                                                   |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>           | Send frame succeed.                                                                                                                                                                                                                               |
| <i>kStatus_ENET_TxFrame-Busy</i> | Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with <i>kStatus-ENET_TxFrameBusy</i> . |

### 8.8.35 **status\_t ENET\_SetTxReclaim ( enet\_handle\_t \* *handle*, bool *isEnabled*, uint8\_t *ringId* )**

## Note

This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

## Parameters

|                  |                                                                                   |
|------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>    | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>isEnabled</i> | Enable or disable flag.                                                           |
| <i>ringId</i>    | The ring index or ring number.                                                    |

## Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Succeed to enable/disable Tx reclaim. |
| <i>kStatus_Fail</i>    | Fail to enable/disable Tx reclaim.    |

### 8.8.36 **void ENET\_ReclaimTxDescriptor ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint8\_t *ringId* )**

This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.



## Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                     |
| <i>handle</i> | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>ringId</i> | The ring index or ring number.                                                    |

### 8.8.37 status\_t ENET\_GetRxFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_rx\_frame\_struct\_t \* *rxFrame*, uint8\_t *ringId* )

This function uses the user-defined allocation and free callbacks. Every time application gets one frame through this function, driver stores the buffer address(es) in enet\_buffer\_struct\_t and allocate new buffer(s) for the BD(s). If there's no memory buffer in the pool, this function drops current one frame to keep the Rx frame in BD ring is as fresh as possible.

## Note

Application must provide a memory pool including at least BD number + n buffers in order for this function to work properly, because each BD must always take one buffer while driver is running, then other extra n buffer(s) can be taken by application. Here n is the ceil(max\_frame\_length(set by RCR) / bd\_rx\_size(set by MRBR)). Application must also provide an array structure in rxFrame->rxBuffArray with n index to receive one complete frame in any case.

## Parameters

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                     |
| <i>handle</i>  | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>rxFrame</i> | The received frame information structure provided by user.                        |
| <i>ringId</i>  | The ring index or ring number.                                                    |

## Return values

|                                   |                                                                 |
|-----------------------------------|-----------------------------------------------------------------|
| <i>kStatus_Success</i>            | Succeed to get one frame and allocate new memory for Rx buffer. |
| <i>kStatus_ENET_RxFrame-Empty</i> | There's no Rx frame in the BD.                                  |

|                                   |                                                       |
|-----------------------------------|-------------------------------------------------------|
| <i>kStatus_ENET_RxFrame-Error</i> | There's issue in this receiving.                      |
| <i>kStatus_ENET_RxFrame-Drop</i>  | There's no new buffer memory for BD, drop this frame. |

### 8.8.38 status\_t ENET\_StartTxFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_tx\_frame\_struct\_t \* *txFrame*, uint8\_t *ringId* )

This function supports scattered buffer transmit, user needs to provide the buffer array.

#### Note

Tx reclaim should be enabled to ensure the Tx buffer ownership can be given back to application after Tx is over.

#### Parameters

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                     |
| <i>handle</i>  | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>txFrame</i> | The Tx frame structure.                                                           |
| <i>ringId</i>  | The ring index or ring number.                                                    |

#### Return values

|                                     |                                                                         |
|-------------------------------------|-------------------------------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to send one frame.                                              |
| <i>kStatus_ENET_TxFrame-Busy</i>    | The BD is not ready for Tx or the reclaim operation still not finishes. |
| <i>kStatus_ENET_TxFrame-OverLen</i> | The Tx frame length is over max ethernet frame length.                  |

### 8.8.39 void ENET\_TransmitIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

#### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |

#### 8.8.40 void ENET\_ReceiveIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |

#### 8.8.41 void ENET\_ErrorIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |

#### 8.8.42 void ENET\_Ptp1588IRQHandler ( ENET\_Type \* *base* )

This is used for the 1588 timer interrupt.

Parameters

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

#### 8.8.43 void ENET\_CommonFrame0IRQHandler ( ENET\_Type \* *base* )

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

Parameters

---

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

## 8.9 Variable Documentation

### 8.9.1 `const clock_ip_name_t s_enetClock[]`

## Chapter 9

# GPC: General Power Controller Driver

### 9.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Power Controller (GPC) module of MCUXpresso SDK devices.

API functions are provided to configure the system about working in dedicated power mode. There are mainly about enabling the power for memory, enabling the wakeup sources for STOP modes, and power up/down operations for various peripherals.

### Macros

- #define `GPC_PGC_TIME_SLOT_TOTAL_NUMBER` `GPC_SLT_CFG_PU_COUNT`  
*Total number of the timeslot.*

### Enumerations

- enum `_gpc_lpm_mode` {  
    `kGPC_RunMode` = 0U,  
    `kGPC_WaitMode` = 1U,  
    `kGPC_StopMode` = 2U }  
*GPC LPM mode definition.*
- enum `_gpc_pgc_ack_sel` {  
    `kGPC_DummyPGCPowerUpAck` = `GPC_PGC_ACK_SEL_DUMMY_PGC_PUP_ACK_MASK`,  
    `kGPC_VirtualPGCPowerUpAck` = `GPC_PGC_ACK_SEL_VIRTUAL_PGC_PUP_ACK_MASK`,  
    `kGPC_DummyPGCPowerDownAck` = `GPC_PGC_ACK_SEL_DUMMY_PGC_PDN_ACK_MASK`,  
    `kGPC_VirtualPGCPowerDownAck` = `GPC_PGC_ACK_SEL_VIRTUAL_PGC_PDN_ACK_MASK`,  
    `kGPC_NocPGCPowerUpAck` = `GPC_PGC_ACK_SEL_NOC_PGC_PUP_ACK`,  
    `kGPC_NocPGCPowerDownAck` = `GPC_PGC_ACK_SEL_NOC_PGC_PDN_ACK` }  
*PGC ack signal selection.*
- enum `_gpc_standby_count` {  
    `kGPC_StandbyCounter4CkilClk` = 0U,  
    `kGPC_StandbyCounter8CkilClk` = 1U,  
    `kGPC_StandbyCounter16CkilClk` = 2U,  
    `kGPC_StandbyCounter32CkilClk` = 3U,  
    `kGPC_StandbyCounter64CkilClk` = 4U,  
    `kGPC_StandbyCounter128CkilClk` = 5U,  
    `kGPC_StandbyCounter256CkilClk` = 6U,  
    `kGPC_StandbyCounter512CkilClk` = 7U }

Standby counter which GPC will wait between PMIC\_STBY\_REQ negation and assertion of PMIC\_READY.

## Functions

- static void [GPC\\_AllowIRQs](#) (GPC\_Type \*base)  
*Allow all the IRQ/Events within the charge of GPC.*
- static void [GPC\\_DisallowIRQs](#) (GPC\_Type \*base)  
*Disallow all the IRQ/Events within the charge of GPC.*
- static uint32\_t [GPC\\_GetLpmMode](#) (GPC\_Type \*base)  
*Get current LPM mode.*
- void [GPC\\_EnableIRQ](#) (GPC\_Type \*base, uint32\_t irqId)  
*Enable the IRQ.*
- void [GPC\\_DisableIRQ](#) (GPC\_Type \*base, uint32\_t irqId)  
*Disable the IRQ.*
- bool [GPC\\_GetIRQStatusFlag](#) (GPC\_Type \*base, uint32\_t irqId)  
*Get the IRQ/Event flag.*
- static void [GPC\\_DsmTriggerMask](#) (GPC\_Type \*base, bool enable)  
*Mask the DSM trigger.*
- static void [GPC\\_WFIMask](#) (GPC\_Type \*base, bool enable)  
*Mask the WFI.*
- static void [GPC\\_SelectPGCAckSignal](#) (GPC\_Type \*base, uint32\_t mask)  
*Select the PGC ACK signal.*
- static void [GPC\\_PowerDownRequestMask](#) (GPC\_Type \*base, bool enable)  
*Power down request to virtual PGC mask or not.*
- static void [GPC\\_PGCMapping](#) (GPC\_Type \*base, uint32\_t mask)  
*PGC CPU Mapping.*
- static void [GPC\\_TimeSlotConfigureForPUS](#) (GPC\_Type \*base, uint8\_t slotIndex, uint32\_t value)  
*Time slot configure.*
- void [GPC\\_EnterWaitMode](#) (GPC\_Type \*base, gpc\_lpm\_config\_t \*config)  
*Enter WAIT mode.*
- void [GPC\\_EnterStopMode](#) (GPC\_Type \*base, gpc\_lpm\_config\_t \*config)  
*Enter STOP mode.*
- void [GPC\\_Init](#) (GPC\_Type \*base, uint32\_t powerUpSlot, uint32\_t powerDownSlot)  
*GPC init function.*

## Driver version

- #define [FSL\\_GPC\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 2, 0))  
*GPC driver version 2.2.0.*

## 9.2 Macro Definition Documentation

### 9.2.1 #define FSL\_GPC\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

## 9.3 Enumeration Type Documentation

### 9.3.1 enum \_gpc\_lpm\_mode

Enumerator

*kGPC\_RunMode* run mode  
*kGPC\_WaitMode* wait mode  
*kGPC\_StopMode* stop mode

### 9.3.2 enum \_gpc\_pgc\_ack\_sel

Enumerator

*kGPC\_DummyPGCPowerUpAck* dummy power up ack signal  
*kGPC\_VirtualPGCPowerUpAck* virtual pgc power up ack signal  
*kGPC\_DummyPGCPowerDownAck* dummy power down ack signal  
*kGPC\_VirtualPGCPowerDownAck* virtual pgc power down ack signal  
*kGPC\_NocPGCPowerUpAck* NOC power up ack signal.  
*kGPC\_NocPGCPowerDownAck* NOC power.

### 9.3.3 enum \_gpc\_standby\_count

Enumerator

*kGPC\_StandbyCounter4CkilClk* 4 ckil clocks  
*kGPC\_StandbyCounter8CkilClk* 8 ckil clocks  
*kGPC\_StandbyCounter16CkilClk* 16 ckil clocks  
*kGPC\_StandbyCounter32CkilClk* 32 ckil clocks  
*kGPC\_StandbyCounter64CkilClk* 64 ckil clocks  
*kGPC\_StandbyCounter128CkilClk* 128 ckil clocks  
*kGPC\_StandbyCounter256CkilClk* 256 ckil clocks  
*kGPC\_StandbyCounter512CkilClk* 512 ckil clocks

## 9.4 Function Documentation

### 9.4.1 static void GPC\_AllowIRQs ( GPC\_Type \* *base* ) [inline], [static]

Parameters

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

#### 9.4.2 static void GPC\_DisallowIRQs ( GPC\_Type \* *base* ) [inline], [static]

Parameters

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

#### 9.4.3 static uint32\_t GPC\_GetLpmMode ( GPC\_Type \* *base* ) [inline], [static]

Parameters

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

Return values

|            |                               |
|------------|-------------------------------|
| <i>lpm</i> | mode, reference _gpc_lpm_mode |
|------------|-------------------------------|

#### 9.4.4 void GPC\_EnableIRQ ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| <i>base</i>  | GPC peripheral base address.                                                                 |
| <i>irqId</i> | ID number of IRQ to be enabled, available range is 0-127,reference SOC headerfile IRQn_Type. |

#### 9.4.5 void GPC\_DisableIRQ ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

---



|              |                                                                                               |
|--------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>  | GPC peripheral base address.                                                                  |
| <i>irqId</i> | ID number of IRQ to be disabled, available range is 0-127,reference SOC headerfile IRQn_Type. |

#### 9.4.6 bool GPC\_GetIRQStatusFlag ( GPC\_Type \* *base*, uint32\_t *irqId* )

Parameters

|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| <i>base</i>  | GPC peripheral base address.                                                                 |
| <i>irqId</i> | ID number of IRQ to be enabled, available range is 0-127,reference SOC headerfile IRQn_Type. |

Returns

Indicated IRQ/Event is asserted or not.

#### 9.4.7 static void GPC\_DsmTriggerMask ( GPC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | GPC peripheral base address.                |
| <i>enable</i> | true to enable mask, false to disable mask. |

#### 9.4.8 static void GPC\_WFIMask ( GPC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | GPC peripheral base address.                |
| <i>enable</i> | true to enable mask, false to disable mask. |

#### 9.4.9 static void GPC\_SelectPGCAckSignal ( GPC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | GPC peripheral base address.              |
| <i>mask</i> | reference <code>_gpc_pgc_ack_sel</code> . |

#### 9.4.10 static void GPC\_PowerDownRequestMask ( GPC\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | GPC peripheral base address.     |
| <i>enable</i> | true to mask, false to not mask. |

#### 9.4.11 static void GPC\_PGCMapping ( GPC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>base</i> | GPC peripheral base address.                     |
| <i>mask</i> | mask value reference PGC CPU mapping definition. |

#### 9.4.12 static void GPC\_TimeSlotConfigureForPUS ( GPC\_Type \* *base*, uint8\_t *slotIndex*, uint32\_t *value* ) [inline], [static]

## Parameters

|                  |                              |
|------------------|------------------------------|
| <i>base</i>      | GPC peripheral base address. |
| <i>slotIndex</i> | time slot index.             |
| <i>value</i>     | value to be configured       |

#### 9.4.13 void GPC\_EnterWaitMode ( GPC\_Type \* *base*, gpc\_lpm\_config\_t \* *config* )

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | GPC peripheral base address. |
| <i>config</i> | lpm mode configurations.     |

**9.4.14 void GPC\_EnterStopMode ( GPC\_Type \* *base*, gpc\_lpm\_config\_t \* *config* )**

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | GPC peripheral base address. |
| <i>config</i> | lpm mode configurations.     |

**9.4.15 void GPC\_Init ( GPC\_Type \* *base*, uint32\_t *powerUpSlot*, uint32\_t *powerDownSlot* )**

## Parameters

|                      |                              |
|----------------------|------------------------------|
| <i>base</i>          | GPC peripheral base address. |
| <i>powerUpSlot</i>   | power up slot number.        |
| <i>powerDownSlot</i> | power down slot number.      |

## Chapter 10

# GPT: General Purpose Timer

### 10.1 Overview

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

### 10.2 Function groups

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

#### 10.2.1 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.

### 10.3 Typical use case

#### 10.3.1 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.*

- 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, 4)`)

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

## Timer Start and Stop

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

## Read the timer period

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

## GPT Input/Output Signal Control

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

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

## GPT Interrupt and Status Interface

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

## Status Interface

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

## 10.4 Data Structure Documentation

### 10.4.1 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.

### Field Documentation

- (1) `gpt_clock_source_t gpt_config_t::clockSource`
- (2) `uint32_t gpt_config_t::divider`
- (3) `bool gpt_config_t::enableFreeRun`
- (4) `bool gpt_config_t::enableRunInWait`
- (5) `bool gpt_config_t::enableRunInStop`
- (6) `bool gpt_config_t::enableRunInDoze`
- (7) `bool gpt_config_t::enableRunInDbg`
- (8) `bool gpt_config_t::enableMode`

## 10.5 Enumeration Type Documentation

### 10.5.1 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.

### 10.5.2 enum `gpt_input_capture_channel_t`

Enumerator

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



### 10.5.3 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.

### 10.5.4 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.

### 10.5.5 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.

### 10.5.6 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.

### 10.5.7 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.

## 10.6 Function Documentation

### 10.6.1 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. |

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

Parameters

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

### 10.6.3 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 = false;
* config->enableMode = true;
*

```

## Parameters

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

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

## Parameters

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

#### 10.6.5 static void GPT\_SetClockSource ( GPT\_Type \* *base*, gpt\_clock\_source\_t *gptClkSource* ) [inline], [static]

## Parameters

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <i>base</i>         | GPT peripheral base address.                                               |
| <i>gptClkSource</i> | Clock source (see <a href="#">gpt_clock_source_t</a> typedef enumeration). |

#### 10.6.6 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).

#### 10.6.7 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).     |

### 10.6.8 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).

### 10.6.9 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).           |

### 10.6.10 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).

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

## Parameters

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

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

## Parameters

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

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

## Parameters

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

## Returns

Current GPT counter value.

**10.6.14 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). |

**10.6.15 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).

**10.6.16** `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.

**10.6.17** `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).   |

**10.6.18** `static gpt_output_operation_mode_t GPT_GetOutputOperationMode (  
GPT_Type * base, gpt_output_compare_channel_t channel ) [inline],  
[static]`

## Parameters

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

## Returns

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

**10.6.19 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.                                                                          |

**10.6.20 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.

**10.6.21 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). |

### 10.6.22 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> |

### 10.6.23 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> |

### 10.6.24 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](#)

### 10.6.25 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.

### 10.6.26 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 11

# GPIO: General-Purpose Input/Output Driver

### 11.1 Overview

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

### 11.2 Typical use case

#### 11.2.1 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, 6)`)  
*GPIO driver version.*

### 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.*
- 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.*

## 11.3 Data Structure Documentation

### 11.3.1 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](#).*

#### Field Documentation

(1) `gpio_pin_direction_t gpio_pin_config_t::direction`

(2) `gpio_interrupt_mode_t gpio_pin_config_t::interruptMode`

## 11.4 Macro Definition Documentation

### 11.4.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 6))

## 11.5 Enumeration Type Documentation

### 11.5.1 enum gpio\_pin\_direction\_t

Enumerator

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

### 11.5.2 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.

## 11.6 Function Documentation

11.6.1 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>Config</i> | pointer to a <a href="#">gpio_pin_config_t</a> structure that contains the configuration information. |

### 11.6.2 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> | GPIO pin output logic level. <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul> |

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

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinWrite](#).

### 11.6.4 static void GPIO\_PortSet ( 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                                          |

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

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortSet](#).

**11.6.6** `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                                          |

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

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortClear](#).

**11.6.8 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                                          |

**11.6.9 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. |
|-------------|-------------------|

**11.6.10 static uint32\_t GPIO\_ReadPinInput ( GPIO\_Type \* *base*, uint32\_t *pin* )**  
**[inline], [static]**

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinRead](#).



**11.6.11** `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. |
|-------------|-----------------------|

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

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinReadPadStatus](#).

**11.6.13** `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-<br/>Mode</i> | pointer to a <a href="#">gpio_interrupt_mode_t</a> structure that contains the interrupt mode information. |

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

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinSetInterruptConfig](#).

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

## Parameters

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

**11.6.16 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. |

**11.6.17 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. |

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

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortDisableInterrupts](#).

**11.6.19 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. |
|----------------|----------------------------|

**11.6.20 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. |
|----------------|----------------------------|

**11.6.21 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. |

**11.6.22 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 12

# I2C: Inter-Integrated Circuit Driver

### 12.1 Overview

#### Modules

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

## 12.2 I2C Driver

### 12.2.1 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.

### 12.2.2 Typical use case

#### 12.2.2.1 Master Operation in functional method

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

#### 12.2.2.2 Master Operation in interrupt transactional method

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

#### 12.2.2.3 Slave Operation in functional method

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

#### 12.2.2.4 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\\_RETRY\\_TIMES](#) 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry 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 {  
[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 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, 7)`)  
*I2C driver version.*

## 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)

- *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.*

## 12.2.3 Data Structure Documentation

### 12.2.3.1 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.*

#### Field Documentation

(1) bool i2c\_master\_config\_t::enableMaster

(2) uint32\_t i2c\_master\_config\_t::baudRate\_Bps

### 12.2.3.2 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.*

### Field Documentation

- (1) `uint32_t i2c_master_transfer_t::flags`
- (2) `uint8_t i2c_master_transfer_t::slaveAddress`
- (3) `i2c_direction_t i2c_master_transfer_t::direction`
- (4) `uint32_t i2c_master_transfer_t::subaddress`

Transferred MSB first.

- (5) `uint8_t i2c_master_transfer_t::subaddressSize`
- (6) `uint8_t* volatile i2c_master_transfer_t::data`
- (7) `volatile size_t i2c_master_transfer_t::dataSize`

### 12.2.3.3 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.*

**Field Documentation**

- (1) `i2c_master_transfer_t i2c_master_handle_t::transfer`
- (2) `size_t i2c_master_handle_t::transferSize`
- (3) `uint8_t i2c_master_handle_t::state`
- (4) `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`
- (5) `void* i2c_master_handle_t::userData`

**12.2.3.4 struct i2c\_slave\_config\_t****Data Fields**

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

**Field Documentation**

- (1) `bool i2c_slave_config_t::enableSlave`
- (2) `uint16_t i2c_slave_config_t::slaveAddress`

**12.2.3.5 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.*

## Field Documentation

(1) `i2c_slave_transfer_event_t i2c_slave_transfer_t::event`

(2) `uint8_t* volatile i2c_slave_transfer_t::data`

(3) `volatile size_t i2c_slave_transfer_t::dataSize`

(4) `status_t i2c_slave_transfer_t::completionStatus`

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

(5) `size_t i2c_slave_transfer_t::transferredCount`

### 12.2.3.6 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.*

## Field Documentation

- (1) `volatile uint8_t i2c_slave_handle_t::state`
- (2) `i2c_slave_transfer_t i2c_slave_handle_t::transfer`
- (3) `uint32_t i2c_slave_handle_t::eventMask`
- (4) `i2c_slave_transfer_callback_t i2c_slave_handle_t::callback`
- (5) `void* i2c_slave_handle_t::userData`

## 12.2.4 Macro Definition Documentation

12.2.4.1 `#define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 7))`

12.2.4.2 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

## 12.2.5 Typedef Documentation

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

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

## 12.2.6 Enumeration Type Documentation

### 12.2.6.1 anonymous enum

Enumerator

*kStatus\_I2C\_Busy* I2C is busy with current transfer.  
*kStatus\_I2C\_Idle* Bus is Idle.  
*kStatus\_I2C\_Nak* NAK received during transfer.  
*kStatus\_I2C\_ArbitrationLost* Arbitration lost during transfer.  
*kStatus\_I2C\_Timeout* Timeout polling status flags.  
*kStatus\_I2C\_Addr\_Nak* NAK received during the address probe.

### 12.2.6.2 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.

### 12.2.6.3 enum \_i2c\_interrupt\_enable

#### Enumerator

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

### 12.2.6.4 enum i2c\_direction\_t

#### Enumerator

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

### 12.2.6.5 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.



### 12.2.6.6 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.

## 12.2.7 Function Documentation

### 12.2.7.1 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            |

**12.2.7.2 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 |
|-------------|------------------|

**12.2.7.3 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. |
|---------------------|--------------------------------------------------|

**12.2.7.4 void I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig* )**

Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

## Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by [I2C\\_SlaveGetDefaultConfig\(\)](#) or it can be custom filled by the user. This is an example.

```
* i2c_slave_config_t config = {
* .enableSlave = true,
* .slaveAddress = 0x1DU,
* };
* I2C_SlaveInit(I2C0, &config);
*
```

## Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | I2C base pointer                               |
| <i>slaveConfig</i> | A pointer to the slave configuration structure |

**12.2.7.5 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 |
|-------------|------------------|

**12.2.7.6 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. |
|--------------------|-------------------------------------------------|

**12.2.7.7 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. |

**12.2.7.8 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.

### 12.2.7.9 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_IntPendingFlag</li> </ul> |

### 12.2.7.10 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.

### 12.2.7.11 static void I2C\_SlaveClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag

## Parameters

|                   |                                                                                                                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | I2C base pointer                                                                                                                                                                                                              |
| <i>statusMask</i> | 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> |

**12.2.7.12 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> |

**12.2.7.13 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> |

**12.2.7.14 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               |

#### 12.2.7.15 **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.                |

#### 12.2.7.16 **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.  |

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

Parameters

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

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>address</i>   | 7-bit slave device address.                   |
| <i>direction</i> | Master transfer directions(transmit/receive). |

Return values

|                         |                                                             |
|-------------------------|-------------------------------------------------------------|
| <i>kStatus_Success</i>  | Successfully send the start signal.                         |
| <i>kStatus_I2C_Busy</i> | Current bus is busy but not occupied by current I2C master. |

#### 12.2.7.18 **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. |

#### 12.2.7.19 **status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize*, uint32\_t *flags* )**

Note

The I2C\_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

## Parameters

|               |                                                                                                                                                       |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | I2C peripheral base pointer.                                                                                                                          |
| <i>rxBuff</i> | The pointer to the data to store the received data.                                                                                                   |
| <i>rxSize</i> | The length in bytes of the data to be received.                                                                                                       |
| <i>flags</i>  | Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop. |

## Return values

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

### 12.2.7.20 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. |

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

## Parameters



|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | I2C peripheral base pointer.                        |
| <i>rxBuff</i> | The pointer to the data to store the received data. |
| <i>rxSize</i> | The length in bytes of the data to be received.     |

#### 12.2.7.22 **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. |

#### 12.2.7.23 **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. |
|-----------------|-------------------------------------------------|

#### 12.2.7.24 **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.      |

#### 12.2.7.25 **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

---

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

#### 12.2.7.26 **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. |

#### 12.2.7.27 **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. |

#### 12.2.7.28 **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.                                   |

#### 12.2.7.29 `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

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

#### 12.2.7.30 `void I2C_SlaveTransferAbort ( I2C_Type * base, i2c_slave_handle_t * handle )`

Note

This API can be called at any time to stop slave for handling the bus events.

## Parameters

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

### 12.2.7.31 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. |

### 12.2.7.32 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 |

## 12.3 I2C FreeRTOS Driver

### 12.3.1 Overview

#### Driver version

- #define `FSL_I2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 7)`)  
*I2C FreeRTOS driver version.*

### 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.*

### 12.3.2 Macro Definition Documentation

#### 12.3.2.1 #define FSL\_I2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 7))

### 12.3.3 Function Documentation

#### 12.3.3.1 `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.

### 12.3.3.2 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. |
|---------------|----------------------|

**12.3.3.3 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.



## 12.4 I2C CMSIS Driver

This section describes the programming interface of the I2C Cortex Microcontroller Software Interface Standard (CMSIS) driver. This driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The I2C CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

### 12.4.1 I2C CMSIS Driver

#### 12.4.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
 if (event == ARM_I2C_EVENT_TRANSFER_DONE)
 {
 g_MasterCompletionFlag = true;
 }
}

/*Init I2C1*/
Driver_I2C1.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C1.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C1.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
Driver_I2C1.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

#### 12.4.1.2 Slave Operation in interrupt transactional method

```
void I2C_SlaveSignalEvent_t(uint32_t event)
{
 /* Transfer done */
 if (event == ARM_I2C_EVENT_TRANSFER_DONE)
 {
 g_SlaveCompletionFlag = true;
 }
}

/*Init I2C1*/
Driver_I2C1.Initialize(I2C_SlaveSignalEvent_t);
```

```
Driver_I2C1.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
Driver_I2C1.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
Driver_I2C1.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;
```

## Chapter 13

# PWM: Pulse Width Modulation Driver

### 13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Pulse Width Modulation (PWM) module of MCUXpresso SDK devices.

### 13.2 PWM Driver

#### 13.2.1 Initialization and deinitialization

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.

### 13.3 Typical use case

#### 13.3.1 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` }  
*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)  
*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.*

## 13.4 Enumeration Type Documentation

### 13.4.1 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.

### 13.4.2 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.

### 13.4.3 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

### 13.4.4 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.

### 13.4.5 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.

### 13.4.6 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.

### 13.4.7 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.

### 13.4.8 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.

### 13.4.9 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

## 13.5 Function Documentation

### 13.5.1 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.

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

Parameters

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

### 13.5.3 void PWM\_GetDefaultConfig ( pwm\_config\_t \* *config* )

The default values are:

```
* config->enableStopMode = false;
* config->enableDozeMode = false;
* config->enableWaitMode = false;
* config->enableDozeMode = false;
* config->clockSource = kPWM_LowFrequencyClock;
* config->prescale = 0U;
* config->outputConfig = kPWM_SetAtRolloverAndClearAtcomparison;
* config->fifoWater = kPWM_FIFOWaterMark_2;
* config->sampleRepeat = kPWM_EachSampleOnce;
* config->byteSwap = kPWM_ByteNoSwap;
* config->halfWordSwap = kPWM_HalfWordNoSwap;
*
```



## Parameters

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

**13.5.4 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 |
|-------------|-----------------------------|

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

## Parameters

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

**13.5.6 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 |
|-------------|-----------------------------|

**13.5.7 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> |

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

Parameters

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

### 13.5.9 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](#)

### 13.5.10 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](#)

**13.5.11** `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> |

### 13.5.12 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](#)

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

## Parameters

|              |                                                                                                                                        |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | PWM peripheral base address                                                                                                            |
| <i>value</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. |

### 13.5.14 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.

**13.5.15** `static void PWM_SetPeriodValue ( PWM_Type * base, uint32_t value )`  
`[inline], [static]`

## Parameters

|              |                                                                                                                                                                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | PWM peripheral base address                                                                                                                                                                                                                      |
| <i>value</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. $PWMO\text{ (Hz)} = PCLK\text{(Hz)} / (\text{period} + 2)$ |

### 13.5.16 `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.

### 13.5.17 `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 14

# UART: Universal Asynchronous Receiver/Transmitter Driver

### 14.1 Overview

#### Modules

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

## 14.2 UART Driver

### 14.2.1 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.

### 14.2.2 Typical use case

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

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



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

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

### 14.2.2.3 UART Receive using the ringbuffer feature

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

### 14.2.2.4 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...](#)

## Macros

- #define [UART\\_RETRY\\_TIMES](#) 0U /\* Defining to zero means to keep waiting for the flag until it is assert/deassert. \*/  
Retry times for waiting flag.

## Typedefs

- typedef void(\* [uart\\_transfer\\_callback\\_t](#) )(UART\_Type \*base, uart\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
UART transfer callback function.

## Enumerations

- enum {
  - kStatus\_UART\_TxBusy = MAKE\_STATUS(kStatusGroup\_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),
  - kStatus\_UART\_Timeout = MAKE\_STATUS(kStatusGroup\_IUART, 15) }

*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 {

```

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

- void \* [s\\_uartHandle](#) []  
*Pointers to uart handles for each instance.*

## Driver version

- #define [FSL\\_UART\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 3, 2))  
*UART driver version.*

## 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.
- status\_t **UART\_WriteBlocking** (UART\_Type \*base, const uint8\_t \*data, size\_t length)  
Writes to the TX register using a blocking method.
- status\_t **UART\_ReadBlocking** (UART\_Type \*base, uint8\_t \*data, size\_t length)  
Read RX data register using a blocking method.

## Transactional

- void **UART\_TransferCreateHandle** (UART\_Type \*base, uart\_handle\_t \*handle, uart\_transfer\_callback\_t callback, void \*userData)  
Initializes the UART handle.
- void **UART\_TransferStartRingBuffer** (UART\_Type \*base, uart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
Sets up the RX ring buffer.
- void **UART\_TransferStopRingBuffer** (UART\_Type \*base, uart\_handle\_t \*handle)  
Aborts the background transfer and uninstalls the ring buffer.
- size\_t **UART\_TransferGetRxRingBufferLength** (uart\_handle\_t \*handle)  
Get the length of received data in RX ring buffer.
- status\_t **UART\_TransferSendNonBlocking** (UART\_Type \*base, uart\_handle\_t \*handle, uart\_transfer\_t \*xfer)  
Transmits a buffer of data using the interrupt method.
- void **UART\_TransferAbortSend** (UART\_Type \*base, uart\_handle\_t \*handle)  
Aborts the interrupt-driven data transmit.
- status\_t **UART\_TransferGetSendCount** (UART\_Type \*base, uart\_handle\_t \*handle, uint32\_t \*count)  
Gets the number of bytes 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, void \*irqHandle)

*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\\_SetRxRTSWatermark](#) (UART\_Type \*base, uint8\_t watermark)  
*This function is used to set the watermark of UART RTS deassertion.*
- 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.*

## 14.2.3 Data Structure Documentation

### 14.2.3.1 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.*
- uint8\_t [rxRTSWatermark](#)  
*RX RTS watermark, RX FIFO data count being larger than this triggers RTS deassertion.*

- bool `enableAutoBaudRate`  
*Enable automatic baud rate detection.*
- bool `enableTx`  
*Enable TX.*
- bool `enableRx`  
*Enable RX.*
- bool `enableRxRTS`  
*RX RTS enable.*
- bool `enableTxCTS`  
*TX CTS enable.*

### Field Documentation

- (1) `uint32_t uart_config_t::baudRate_Bps`
- (2) `uart_parity_mode_t uart_config_t::parityMode`
- (3) `uart_stop_bit_count_t uart_config_t::stopBitCount`

### 14.2.3.2 struct `uart_transfer_t`

#### Data Fields

- `size_t dataSize`  
*The byte count to be transfer.*
- `uint8_t * data`  
*The buffer of data to be transfer.*
- `uint8_t * rxData`  
*The buffer to receive data.*
- `const uint8_t * txData`  
*The buffer of data to be sent.*

### Field Documentation

- (1) `uint8_t* uart_transfer_t::data`
- (2) `uint8_t* uart_transfer_t::rxData`
- (3) `const uint8_t* uart_transfer_t::txData`
- (4) `size_t uart_transfer_t::dataSize`

### 14.2.3.3 struct `_uart_handle`

Forward declaration of the handle typedef.

#### Data Fields

- `const uint8_t *volatile txData`  
*Address of remaining data to send.*

- volatile size\_t [txDataSize](#)  
*Size of the remaining data to send.*
- size\_t [txDataSizeAll](#)  
*Size of the data to send out.*
- uint8\_t \*volatile [rxData](#)  
*Address of remaining data to receive.*
- volatile size\_t [rxDataSize](#)  
*Size of the remaining data to receive.*
- size\_t [rxDataSizeAll](#)  
*Size of the data to receive.*
- uint8\_t \* [rxRingBuffer](#)  
*Start address of the receiver ring buffer.*
- size\_t [rxRingBufferSize](#)  
*Size of the ring buffer.*
- volatile uint16\_t [rxRingBufferHead](#)  
*Index for the driver to store received data into ring buffer.*
- volatile uint16\_t [rxRingBufferTail](#)  
*Index for the user to get data from the ring buffer.*
- [uart\\_transfer\\_callback\\_t](#) [callback](#)  
*Callback function.*
- void \* [userData](#)  
*UART callback function parameter.*
- volatile uint8\_t [txState](#)  
*TX transfer state.*
- volatile uint8\_t [rxState](#)  
*RX transfer state.*



## Field Documentation

- (1) `const uint8_t* volatile uart_handle_t::txData`
- (2) `volatile size_t uart_handle_t::txDataSize`
- (3) `size_t uart_handle_t::txDataSizeAll`
- (4) `uint8_t* volatile uart_handle_t::rxData`
- (5) `volatile size_t uart_handle_t::rxDataSize`
- (6) `size_t uart_handle_t::rxDataSizeAll`
- (7) `uint8_t* uart_handle_t::rxRingBuffer`
- (8) `size_t uart_handle_t::rxRingBufferSize`
- (9) `volatile uint16_t uart_handle_t::rxRingBufferHead`
- (10) `volatile uint16_t uart_handle_t::rxRingBufferTail`
- (11) `uart_transfer_callback_t uart_handle_t::callback`
- (12) `void* uart_handle_t::userData`
- (13) `volatile uint8_t uart_handle_t::txState`

## 14.2.4 Macro Definition Documentation

14.2.4.1 `#define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`

14.2.4.2 `#define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */`

## 14.2.5 Typedef Documentation

14.2.5.1 `typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)`

## 14.2.6 Enumeration Type Documentation

### 14.2.6.1 anonymous enum

Enumerator

*kStatus\_UART\_TxBusy* Transmitter is busy.  
*kStatus\_UART\_RxBusy* Receiver is busy.

*kStatus\_UART\_TxIdle* UART transmitter is idle.  
*kStatus\_UART\_RxIdle* UART receiver is idle.  
*kStatus\_UART\_TxWatermarkTooLarge* TX FIFO watermark too large.  
*kStatus\_UART\_RxWatermarkTooLarge* RX FIFO watermark too large.  
*kStatus\_UART\_FlagCannotClearManually* UART flag can't be manually cleared.  
*kStatus\_UART\_Error* Error happens on UART.  
*kStatus\_UART\_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.  
*kStatus\_UART\_Timeout* UART times out.

#### 14.2.6.2 enum uart\_data\_bits\_t

Enumerator

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

#### 14.2.6.3 enum uart\_parity\_mode\_t

Enumerator

*kUART\_ParityDisabled* Parity disabled.  
*kUART\_ParityEven* Even error check is selected.  
*kUART\_ParityOdd* Odd error check is selected.

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

Enumerator

*kUART\_OneStopBit* One stop bit.  
*kUART\_TwoStopBit* Two stop bits.

#### 14.2.6.5 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.

#### 14.2.6.6 enum \_uart\_interrupt\_enable

#### 14.2.6.7 anonymous enum

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.

## 14.2.7 Function Documentation

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

Parameters

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

Returns

UART instance.

### 14.2.7.2 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, UBMR, UBRC , URXD, UTXD and UTS[6-3]

Parameters

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

### 14.2.7.3 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\\_GetDefault-Config\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the 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 |
|------------------------|-------------------------|

**14.2.7.4 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. |
|-------------|-------------------------------|

**14.2.7.5 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. |
|---------------|---------------------------------------|

**14.2.7.6 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.                              |

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

## Parameters

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

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

## Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>base</i>      | UART base pointer.                                                                       |
| <i>condition</i> | IDLE line detect condition of the enumerators in <a href="#">uart_idle_condition_t</a> . |

**14.2.7.9 static void UART\_Disable ( UART\_Type \* *base* ) [inline], [static]**

## Parameters

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

**14.2.7.10 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. |
|----------------|-------------------------------------|

**14.2.7.11 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. |

**14.2.7.12 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> . |

**14.2.7.13 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> . |

**14.2.7.14 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](#).

**14.2.7.15 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. |

**14.2.7.16 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. |

#### 14.2.7.17 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. |

#### 14.2.7.18 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.

#### 14.2.7.19 status\_t UART\_WriteBlocking ( UART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | UART peripheral base address.       |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                             |                                         |
|-----------------------------|-----------------------------------------|
| <i>kStatus_UART_Timeout</i> | Transmission timed out and was aborted. |
| <i>kStatus_Success</i>      | Successfully wrote all data.            |

#### 14.2.7.20 **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_UART_Timeout</i>            | Transmission timed out and was aborted.         |
| <i>kStatus_Success</i>                 | Successfully received all data.                 |

#### 14.2.7.21 **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. |

#### 14.2.7.22 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.                                                                         |

#### 14.2.7.23 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.          |

#### 14.2.7.24 **size\_t UART\_TransferGetRxRingBufferLength ( uart\_handle\_t \* *handle* )**

Parameters

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

Returns

Length of received data in RX ring buffer.

#### 14.2.7.25 **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.                                                                  |

#### 14.2.7.26 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.          |

#### 14.2.7.27 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> ; |

#### 14.2.7.28 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.

## 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.                                    |

#### 14.2.7.29 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.          |

#### 14.2.7.30 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 <code>count</code> ; |

#### 14.2.7.31 void UART\_TransferHandleIRQ ( UART\_Type \* *base*, void \* *irqHandle* )

This function handles the UART transmit and receive IRQ request.

## Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | UART peripheral base address. |
| <i>irqHandle</i> | UART handle pointer.          |

#### 14.2.7.32 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. |

#### 14.2.7.33 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 RxFIFO. The fill level in the RxFIFO at which a DMA request is generated is controlled by the RXTL bits .

## Parameters

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

#### 14.2.7.34 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. |

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

The RTS signal deasserts whenever the data count in RxFIFO reaches the Rx RTS watermark.

## Parameters

|                  |                       |
|------------------|-----------------------|
| <i>base</i>      | UART base pointer.    |
| <i>watermark</i> | The Rx RTS watermark. |

#### 14.2.7.36 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. |
|------------------|------------------------|

#### 14.2.7.37 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> |

#### 14.2.7.38 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.

### 14.2.8 Variable Documentation

#### 14.2.8.1 void\* s\_uartHandle[]

## 14.3 UART FreeRTOS Driver

### 14.3.1 Overview

#### Data Structures

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

#### Driver version

- #define `FSL_UART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)  
*UART FreeRTOS driver version 2.1.1.*

#### UART RTOS Operation

- int `UART_RTOS_Init` (`uart_rtos_handle_t *handle`, `uart_handle_t *t_handle`, const `uart_rtos_config_t *cfg`)  
*Initializes a UART instance for operation in RTOS.*
- int `UART_RTOS_Deinit` (`uart_rtos_handle_t *handle`)  
*Deinitializes a UART instance for operation.*

#### UART transactional Operation

- int `UART_RTOS_Send` (`uart_rtos_handle_t *handle`, `uint8_t *buffer`, `uint32_t length`)  
*Sends data in the background.*
- int `UART_RTOS_Receive` (`uart_rtos_handle_t *handle`, `uint8_t *buffer`, `uint32_t length`, `size_t *received`)  
*Receives data.*

### 14.3.2 Data Structure Documentation

#### 14.3.2.1 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_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.*

### 14.3.3 Macro Definition Documentation

#### 14.3.3.1 `#define FSL_UART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

### 14.3.4 Function Documentation

#### 14.3.4.1 `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.

#### 14.3.4.2 `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. |
|---------------|-----------------------|

#### 14.3.4.3 `int UART_RTOS_Send ( uart_rtos_handle_t * handle, uint8_t * buffer, uint32_t length )`

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>handle</i> | The RTOS UART handle.              |
| <i>buffer</i> | The pointer to the buffer to send. |
| <i>length</i> | The number of bytes to send.       |

#### 14.3.4.4 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. |

## 14.4 UART SDMA Driver

### 14.4.1 Overview

#### Data Structures

- struct `uart_sdma_handle_t`  
*UART sDMA handle. [More...](#)*

#### Typedefs

- typedef void(\* `uart_sdma_transfer_callback_t`)(UART\_Type \*base, uart\_sdma\_handle\_t \*handle, `status_t` status, void \*userData)  
*UART transfer callback function.*

#### Driver version

- #define `FSL_UART_SDMA_DRIVER_VERSION` (`MAKE_VERSION`(2, 3, 0))  
*UART SDMA driver version.*

#### sDMA transactional

- void `UART_TransferCreateHandleSDMA` (UART\_Type \*base, uart\_sdma\_handle\_t \*handle, `uart_sdma_transfer_callback_t` callback, void \*userData, `sdma_handle_t` \*txSdmaHandle, `sdma_handle_t` \*rxSdmaHandle, uint32\_t eventSourceTx, uint32\_t eventSourceRx)  
*Initializes the UART handle which is used in transactional functions.*
- `status_t` `UART_SendSDMA` (UART\_Type \*base, uart\_sdma\_handle\_t \*handle, `uart_transfer_t` \*xfer)  
*Sends data using sDMA.*
- `status_t` `UART_ReceiveSDMA` (UART\_Type \*base, uart\_sdma\_handle\_t \*handle, `uart_transfer_t` \*xfer)  
*Receives data using sDMA.*
- void `UART_TransferAbortSendSDMA` (UART\_Type \*base, uart\_sdma\_handle\_t \*handle)  
*Aborts the sent data using sDMA.*
- void `UART_TransferAbortReceiveSDMA` (UART\_Type \*base, uart\_sdma\_handle\_t \*handle)  
*Aborts the receive data using sDMA.*
- void `UART_TransferSdmaHandleIRQ` (UART\_Type \*base, void \*uartSdmaHandle)  
*UART IRQ handle function.*

## 14.4.2 Data Structure Documentation

### 14.4.2.1 struct \_uart\_sdma\_handle

#### Data Fields

- [uart\\_sdma\\_transfer\\_callback\\_t](#) *callback*  
*Callback function.*
- void \* [userData](#)  
*UART callback function parameter.*
- size\_t [rxDataSizeAll](#)  
*Size of the data to receive.*
- size\_t [txDataSizeAll](#)  
*Size of the data to send out.*
- [sdma\\_handle\\_t](#) \* [txSdmaHandle](#)  
*The sDMA TX channel used.*
- [sdma\\_handle\\_t](#) \* [rxSdmaHandle](#)  
*The sDMA RX channel used.*
- volatile uint8\_t [txState](#)  
*TX transfer state.*
- volatile uint8\_t [rxState](#)  
*RX transfer state.*

## Field Documentation

- (1) `uart_sdma_transfer_callback_t` `uart_sdma_handle_t::callback`
- (2) `void*` `uart_sdma_handle_t::userData`
- (3) `size_t` `uart_sdma_handle_t::rxDataSizeAll`
- (4) `size_t` `uart_sdma_handle_t::txDataSizeAll`
- (5) `sdma_handle_t*` `uart_sdma_handle_t::txSdmaHandle`
- (6) `sdma_handle_t*` `uart_sdma_handle_t::rxSdmaHandle`
- (7) `volatile uint8_t` `uart_sdma_handle_t::txState`

## 14.4.3 Macro Definition Documentation

14.4.3.1 `#define FSL_UART_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`

## 14.4.4 Typedef Documentation

14.4.4.1 `typedef void(* uart_sdma_transfer_callback_t)(UART_Type *base, uart_sdma_handle_t *handle, status_t status, void *userData)`

## 14.4.5 Function Documentation

14.4.5.1 `void UART_TransferCreateHandleSDMA ( UART_Type * base, uart_sdma_handle_t * handle, uart_sdma_transfer_callback_t callback, void * userData, sdma_handle_t * txSdmaHandle, sdma_handle_t * rxSdmaHandle, uint32_t eventSourceTx, uint32_t eventSourceRx )`

## Parameters

|                      |                                                           |
|----------------------|-----------------------------------------------------------|
| <i>base</i>          | UART peripheral base address.                             |
| <i>handle</i>        | Pointer to the <code>uart_sdma_handle_t</code> structure. |
| <i>callback</i>      | UART callback, NULL means no callback.                    |
| <i>userData</i>      | User callback function data.                              |
| <i>rxSdmaHandle</i>  | User-requested DMA handle for RX DMA transfer.            |
| <i>txSdmaHandle</i>  | User-requested DMA handle for TX DMA transfer.            |
| <i>eventSourceTx</i> | Eventsource for TX DMA transfer.                          |
| <i>eventSourceRx</i> | Eventsource for RX DMA transfer.                          |

#### 14.4.5.2 **status\_t UART\_SendSDMA ( UART\_Type \* *base*, uart\_sdma\_handle\_t \* *handle*, uart\_transfer\_t \* *xfer* )**

This function sends data using sDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

## Parameters

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

## Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | if succeeded; otherwise failed. |
| <i>kStatus_UART_TxBusy</i>     | Previous transfer ongoing.      |
| <i>kStatus_InvalidArgument</i> | Invalid argument.               |

#### 14.4.5.3 **status\_t UART\_ReceiveSDMA ( UART\_Type \* *base*, uart\_sdma\_handle\_t \* *handle*, uart\_transfer\_t \* *xfer* )**

This function receives data using sDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.



## Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | UART peripheral base address.                                       |
| <i>handle</i> | Pointer to the <code>uart_sdma_handle_t</code> structure.           |
| <i>xfer</i>   | UART sDMA transfer structure. See <a href="#">uart_transfer_t</a> . |

## Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | if succeeded; otherwise failed. |
| <i>kStatus_UART_RxBusy</i>     | Previous transfer ongoing.      |
| <i>kStatus_InvalidArgument</i> | Invalid argument.               |

#### 14.4.5.4 void UART\_TransferAbortSendSDMA ( UART\_Type \* *base*, `uart_sdma_handle_t` \* *handle* )

This function aborts sent data using sDMA.

## Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | UART peripheral base address.                             |
| <i>handle</i> | Pointer to the <code>uart_sdma_handle_t</code> structure. |

#### 14.4.5.5 void UART\_TransferAbortReceiveSDMA ( UART\_Type \* *base*, `uart_sdma_handle_t` \* *handle* )

This function aborts receive data using sDMA.

## Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | UART peripheral base address.                             |
| <i>handle</i> | Pointer to the <code>uart_sdma_handle_t</code> structure. |

#### 14.4.5.6 void UART\_TransferSdmaHandleIRQ ( UART\_Type \* *base*, void \* *uartSdmaHandle* )

This function handles the UART transmit complete IRQ request and invoke user callback.

## Parameters

|                        |                               |
|------------------------|-------------------------------|
| <i>base</i>            | UART peripheral base address. |
| <i>uartSdma-Handle</i> | UART handle pointer.          |

## 14.5 UART CMSIS Driver

This section describes the programming interface of the UART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The UART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

### 14.5.1 Function groups

#### 14.5.1.1 UART CMSIS GetVersion Operation

This function group will return the UART CMSIS Driver version to user.

#### 14.5.1.2 UART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 14.5.1.3 UART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the uart instance . And this API must be called before you configure an uart instance or after you Deinit an uart instance. The right steps to start an instance is that you must initialize the instance which been selected firstly, then you can power on the instance. After these all have been done, you can configure the instance by using control operation. If you want to Uninitialize the instance, you must power off the instance first.

#### 14.5.1.4 UART CMSIS Transfer Operation

This function group controls the transfer, send/receive data.

#### 14.5.1.5 UART CMSIS Status Operation

This function group gets the UART transfer status.

#### 14.5.1.6 UART CMSIS Control Operation

This function can configure an instance ,set baudrate for uart, get current baudrate ,set transfer data bits and other control command.

# Chapter 15

## MU: Messaging Unit

### 15.1 Overview

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

### 15.2 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.

#### 15.2.1 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.

#### 15.2.2 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.

### 15.2.3 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.

### 15.2.4 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.

### 15.2.5 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` = (int)(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.*
- enum `mu_msg_reg_index_t`  
*MU message register.*

## Driver version

- #define `FSL_MU_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 2))  
*MU driver version.*

## 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 uint32\_t [MU\\_GetInterruptsPending](#) (MU\_Type \*base)  
*Gets the MU IRQ pending status.*
- 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.

## 15.3 Macro Definition Documentation

### 15.3.1 #define FSL\_MU\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))

## 15.4 Enumeration Type Documentation

### 15.4.1 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 1 pending.  
***kMU\_GenInt2Flag*** General purpose interrupt 2 pending.  
***kMU\_GenInt3Flag*** General purpose interrupt 3 pending.  
***kMU\_EventPendingFlag*** MU event pending.  
***kMU\_FlagsUpdatingFlag*** MU flags update is on-going.

### 15.4.2 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.  
*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.

### 15.4.3 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.

## 15.5 Function Documentation

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

This function enables the MU clock only.

Parameters

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

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

This function disables the MU clock only.

Parameters

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

### 15.5.3 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, kMU_MsgReg0, MSG_VAL); Write message to the TX0
 register.
*
```

#### Parameters

|                 |                                                             |
|-----------------|-------------------------------------------------------------|
| <i>base</i>     | MU peripheral base address.                                 |
| <i>regIndex</i> | TX register index, see <a href="#">mu_msg_reg_index_t</a> . |
| <i>msg</i>      | Message to send.                                            |

### 15.5.4 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> | MU message register, see <a href="#">mu_msg_reg_index_t</a> |
| <i>msg</i>      | Message to send.                                            |

### 15.5.5 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, kMU_MsgReg0); Read message from RX0
 register.
*
```

## Parameters

|             |                                                          |
|-------------|----------------------------------------------------------|
| <i>base</i> | MU peripheral base address.                              |
| <i>RX</i>   | register index, see <a href="#">mu_msg_reg_index_t</a> . |

## Returns

The received message.

### 15.5.6 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> | MU message register, see <a href="#">mu_msg_reg_index_t</a> |

## Returns

The received message.

### 15.5.7 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.  |

**15.5.8 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.  |

**15.5.9 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.

**15.5.10 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, kMU_MsgReg0, MSG0_VAL);
* }
```

```

* if (kMU_Tx1EmptyFlag & flags)
* {
* The TX1 register is empty. Message can be sent.
* MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
* }
*

```

## Parameters

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

## Returns

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

### 15.5.11 `static uint32_t MU_GetInterruptsPending ( MU_Type * base ) [inline], [static]`

This function returns the bit mask of the pending MU IRQs.

## Parameters

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

## Returns

Bit mask of the MU IRQs pending.

### 15.5.12 `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>• <code>kMU_Tx0EmptyFlag</code></li> <li>• <code>kMU_Tx1EmptyFlag</code></li> <li>• <code>kMU_Tx2EmptyFlag</code></li> <li>• <code>kMU_Tx3EmptyFlag</code></li> <li>• <code>kMU_Rx0FullFlag</code></li> <li>• <code>kMU_Rx1FullFlag</code></li> <li>• <code>kMU_Rx2FullFlag</code></li> <li>• <code>kMU_Rx3FullFlag</code></li> <li>• <code>kMU_EventPendingFlag</code></li> <li>• <code>kMU_FlagsUpdatingFlag</code></li> <li>• <code>kMU_OtherSideInResetFlag</code></li> </ul> |

### 15.5.13 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> . |

### 15.5.14 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> . |

**15.5.15 `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.
* }
*
```

## 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.  |

**15.5.16 `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. |
|-------------|-----------------------------|

**15.5.17 `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.

### 15.5.18 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.

### 15.5.19 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.   |

### 15.5.20 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. |
|-------------|-----------------------------|

### 15.5.21 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

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

## Note

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

### 15.5.22 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\_Mask-HardwareReset. 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 <i>holdReset</i> is true, this parameter is useless.                                                                                                                                                                                                                    |

### 15.5.23 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. |

### 15.5.24 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 16

# PDM: Microphone Interface

### 16.1 Overview

#### Modules

- [PDM Driver](#)
- [PDM SDMA Driver](#)

### ***16.2 Typical use case***

## 16.3 PDM Driver

### 16.3.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Microphone Interface (PDM) module of MCUXpresso SDK devices.

PDM driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for PDM initialization, configuration, and operation for the optimization and customization purpose. Using the functional API requires the knowledge of the PDM peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. PDM 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. Initialize the handle by calling the [PDM\\_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [PDM\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with `kStatus_PDM_Idle` status.

### 16.3.2 Typical use case

#### 16.3.2.1 PDM receive using an interrupt method

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

#### 16.3.2.2 PDM receive using a SDMA method

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

#### 16.3.2.3 PDM receive using a EDMA method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pdm/pdm-_edma_transfer` Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pdm/pdm_sai_edma` Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pdm/pdm_sai_multi_channel_edma`

### 16.3.2.4 PDM receive using a transactional method

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pdm/pdm-  
\_interrupt\_transfer

## Data Structures

- struct [pdm\\_channel\\_config\\_t](#)  
*PDM channel configurations. [More...](#)*
- struct [pdm\\_config\\_t](#)  
*PDM user configuration structure. [More...](#)*
- struct [pdm\\_hwvad\\_config\\_t](#)  
*PDM voice activity detector user configuration structure. [More...](#)*
- struct [pdm\\_hwvad\\_noise\\_filter\\_t](#)  
*PDM voice activity detector noise filter user configuration structure. [More...](#)*
- struct [pdm\\_hwvad\\_zero\\_cross\\_detector\\_t](#)  
*PDM voice activity detector zero cross detector configuration structure. [More...](#)*
- struct [pdm\\_transfer\\_t](#)  
*PDM SDMA transfer structure. [More...](#)*
- struct [pdm\\_hwvad\\_notification\\_t](#)  
*PDM HWVAD notification structure. [More...](#)*
- struct [pdm\\_handle\\_t](#)  
*PDM handle structure. [More...](#)*

## Macros

- #define [PDM\\_XFER\\_QUEUE\\_SIZE](#) (4U)  
*PDM XFER QUEUE SIZE.*

## Typedefs

- typedef void(\* [pdm\\_transfer\\_callback\\_t](#))(PDM\_Type \*base, pdm\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*PDM transfer callback prototype.*
- typedef void(\* [pdm\\_hwvad\\_callback\\_t](#))([status\\_t](#) status, void \*userData)  
*PDM HWVAD callback prototype.*

## Enumerations

- enum {
  - kStatus\_PDM\_Busy = MAKE\_STATUS(kStatusGroup\_PDM, 0),
  - kStatus\_PDM\_FIFO\_ERROR = MAKE\_STATUS(kStatusGroup\_PDM, 2),
  - kStatus\_PDM\_QueueFull = MAKE\_STATUS(kStatusGroup\_PDM, 3),
  - kStatus\_PDM\_Idle = MAKE\_STATUS(kStatusGroup\_PDM, 4),
  - kStatus\_PDM\_Output\_ERROR = MAKE\_STATUS(kStatusGroup\_PDM, 5),
  - kStatus\_PDM\_ChannelConfig\_Failed = MAKE\_STATUS(kStatusGroup\_PDM, 6),
  - kStatus\_PDM\_HWVAD\_VoiceDetected = MAKE\_STATUS(kStatusGroup\_PDM, 7),
  - kStatus\_PDM\_HWVAD\_Error = MAKE\_STATUS(kStatusGroup\_PDM, 8) }

*PDM return status.*
- enum \_pdm\_interrupt\_enable {
  - kPDM\_ErrorInterruptEnable = PDM\_CTRL\_1\_ERREN\_MASK,
  - kPDM\_FIFOInterruptEnable = PDM\_CTRL\_1\_DISEL(2U) }

*The PDM interrupt enable flag.*
- enum \_pdm\_internal\_status {
  - kPDM\_StatusDfBusyFlag = (int)PDM\_STAT\_BSY\_FIL\_MASK,
  - kPDM\_StatusFIRFilterReady = PDM\_STAT\_FIR\_RDY\_MASK,
  - kPDM\_StatusCh0FifoDataAvaliable = PDM\_STAT\_CH0F\_MASK,
  - kPDM\_StatusCh1FifoDataAvaliable = PDM\_STAT\_CH1F\_MASK,
  - kPDM\_StatusCh2FifoDataAvaliable = PDM\_STAT\_CH2F\_MASK,
  - kPDM\_StatusCh3FifoDataAvaliable = PDM\_STAT\_CH3F\_MASK,
  - kPDM\_StatusCh4FifoDataAvaliable = PDM\_STAT\_CH4F\_MASK,
  - kPDM\_StatusCh5FifoDataAvaliable = PDM\_STAT\_CH5F\_MASK,
  - kPDM\_StatusCh6FifoDataAvaliable = PDM\_STAT\_CH6F\_MASK,
  - kPDM\_StatusCh7FifoDataAvaliable = PDM\_STAT\_CH7F\_MASK }

*The PDM status.*
- enum \_pdm\_channel\_enable\_mask {
  - kPDM\_EnableChannel0 = PDM\_STAT\_CH0F\_MASK,
  - kPDM\_EnableChannel1 = PDM\_STAT\_CH1F\_MASK,
  - kPDM\_EnableChannel2 = PDM\_STAT\_CH2F\_MASK,
  - kPDM\_EnableChannel3 = PDM\_STAT\_CH3F\_MASK,
  - kPDM\_EnableChannel4 = PDM\_STAT\_CH4F\_MASK,
  - kPDM\_EnableChannel5 = PDM\_STAT\_CH5F\_MASK,
  - kPDM\_EnableChannel6 = PDM\_STAT\_CH6F\_MASK,
  - kPDM\_EnableChannel7 = PDM\_STAT\_CH7F\_MASK }

*PDM channel enable mask.*
- enum \_pdm\_fifo\_status {

```

kPDM_FifoStatusUnderflowCh0 = PDM_FIFO_STAT_FIFOUND0_MASK,
kPDM_FifoStatusUnderflowCh1 = PDM_FIFO_STAT_FIFOUND1_MASK,
kPDM_FifoStatusUnderflowCh2 = PDM_FIFO_STAT_FIFOUND2_MASK,
kPDM_FifoStatusUnderflowCh3 = PDM_FIFO_STAT_FIFOUND3_MASK,
kPDM_FifoStatusUnderflowCh4 = PDM_FIFO_STAT_FIFOUND4_MASK,
kPDM_FifoStatusUnderflowCh5 = PDM_FIFO_STAT_FIFOUND5_MASK,
kPDM_FifoStatusUnderflowCh6 = PDM_FIFO_STAT_FIFOUND6_MASK,
kPDM_FifoStatusUnderflowCh7 = PDM_FIFO_STAT_FIFOUND6_MASK,
kPDM_FifoStatusOverflowCh0 = PDM_FIFO_STAT_FIFOOVF0_MASK,
kPDM_FifoStatusOverflowCh1 = PDM_FIFO_STAT_FIFOOVF1_MASK,
kPDM_FifoStatusOverflowCh2 = PDM_FIFO_STAT_FIFOOVF2_MASK,
kPDM_FifoStatusOverflowCh3 = PDM_FIFO_STAT_FIFOOVF3_MASK,
kPDM_FifoStatusOverflowCh4 = PDM_FIFO_STAT_FIFOOVF4_MASK,
kPDM_FifoStatusOverflowCh5 = PDM_FIFO_STAT_FIFOOVF5_MASK,
kPDM_FifoStatusOverflowCh6 = PDM_FIFO_STAT_FIFOOVF6_MASK,
kPDM_FifoStatusOverflowCh7 = PDM_FIFO_STAT_FIFOOVF7_MASK }

```

*The PDM fifo status.*

- enum `_pdm_output_status` {
 

```

kPDM_OutputStatusUnderFlowCh0 = PDM_OUT_STAT_OUTUNF0_MASK,
kPDM_OutputStatusUnderFlowCh1 = PDM_OUT_STAT_OUTUNF1_MASK,
kPDM_OutputStatusUnderFlowCh2 = PDM_OUT_STAT_OUTUNF2_MASK,
kPDM_OutputStatusUnderFlowCh3 = PDM_OUT_STAT_OUTUNF3_MASK,
kPDM_OutputStatusUnderFlowCh4 = PDM_OUT_STAT_OUTUNF4_MASK,
kPDM_OutputStatusUnderFlowCh5 = PDM_OUT_STAT_OUTUNF5_MASK,
kPDM_OutputStatusUnderFlowCh6 = PDM_OUT_STAT_OUTUNF6_MASK,
kPDM_OutputStatusUnderFlowCh7 = PDM_OUT_STAT_OUTUNF7_MASK,
kPDM_OutputStatusOverFlowCh0 = PDM_OUT_STAT_OUTOVF0_MASK,
kPDM_OutputStatusOverFlowCh1 = PDM_OUT_STAT_OUTOVF1_MASK,
kPDM_OutputStatusOverFlowCh2 = PDM_OUT_STAT_OUTOVF2_MASK,
kPDM_OutputStatusOverFlowCh3 = PDM_OUT_STAT_OUTOVF3_MASK,
kPDM_OutputStatusOverFlowCh4 = PDM_OUT_STAT_OUTOVF4_MASK,
kPDM_OutputStatusOverFlowCh5 = PDM_OUT_STAT_OUTOVF5_MASK,
kPDM_OutputStatusOverFlowCh6 = PDM_OUT_STAT_OUTOVF6_MASK,
kPDM_OutputStatusOverFlowCh7 = PDM_OUT_STAT_OUTOVF7_MASK }

```

*The PDM output status.*

- enum `pdm_dc_removal_t` {
 

```

kPDM_DcRemoverCutOff21Hz = 0U,
kPDM_DcRemoverCutOff83Hz = 1U,
kPDM_DcRemoverCutOff152Hz = 2U,
kPDM_DcRemoverBypass = 3U }

```

*PDM DC remover configurations.*

- enum `pdm_df_quality_mode_t` {



```

kPDM_QualityModeMedium = 0U,
kPDM_QualityModeHigh = 1U,
kPDM_QualityModeLow = 7U,
kPDM_QualityModeVeryLow0 = 6U,
kPDM_QualityModeVeryLow1 = 5U,
kPDM_QualityModeVeryLow2 = 4U }

```

*PDM decimation filter quality mode.*

- enum `_pdm_qulaity_mode_k_factor` {  
`kPDM_QualityModeHighKFactor` = 1U,  
`kPDM_QualityModeMediumKFactor` = 2U,  
`kPDM_QualityModeLowKFactor` = 4U,  
`kPDM_QualityModeVeryLow2KFactor` = 8U }

*PDM quality mode K factor.*

- enum `pdm_df_output_gain_t` {  
`kPDM_DfOutputGain0` = 0U,  
`kPDM_DfOutputGain1` = 1U,  
`kPDM_DfOutputGain2` = 2U,  
`kPDM_DfOutputGain3` = 3U,  
`kPDM_DfOutputGain4` = 4U,  
`kPDM_DfOutputGain5` = 5U,  
`kPDM_DfOutputGain6` = 6U,  
`kPDM_DfOutputGain7` = 7U,  
`kPDM_DfOutputGain8` = 8U,  
`kPDM_DfOutputGain9` = 9U,  
`kPDM_DfOutputGain10` = 0xAU,  
`kPDM_DfOutputGain11` = 0xBU,  
`kPDM_DfOutputGain12` = 0xCU,  
`kPDM_DfOutputGain13` = 0xDU,  
`kPDM_DfOutputGain14` = 0xEU,  
`kPDM_DfOutputGain15` = 0xFU }

*PDM decimation filter output gain.*

- enum `_pdm_data_width` { `kPDM_DataWidth16` = 2U }

*PDM data width.*

- enum `_pdm_hwvad_interrupt_enable` {  
`kPDM_HwvadErrorInterruptEnable` = PDM\_VAD0\_CTRL\_1\_VADERIE\_MASK,  
`kPDM_HwvadInterruptEnable` = PDM\_VAD0\_CTRL\_1\_VADIE\_MASK }

*PDM voice activity detector interrupt type.*

- enum `_pdm_hwvad_int_status` {  
`kPDM_HwvadStatusInputSaturation` = PDM\_VAD0\_STAT\_VADINSATF\_MASK,  
`kPDM_HwvadStatusVoiceDetectFlag` = PDM\_VAD0\_STAT\_VADIF\_MASK }

*The PDM hwvad interrupt status flag.*

- enum `pdm_hwvad_hpf_config_t` {  
`kPDM_HwvadHpfBypassed` = 0x0U,  
`kPDM_HwvadHpfCutOffFreq1750Hz` = 0x1U,  
`kPDM_HwvadHpfCutOffFreq215Hz` = 0x2U,  
`kPDM_HwvadHpfCutOffFreq102Hz` = 0x3U }

- *High pass filter configure cut-off frequency.*
- enum `pdm_hwvad_filter_status_t` {  
`kPDM_HwvadInternalFilterNormalOperation` = 0U,  
`kPDM_HwvadInternalFilterInitial` = PDM\_VAD0\_CTRL\_1\_VADST10\_MASK }  
*HWVAD internal filter status.*
- enum `pdm_hwvad_zcd_result_t` {  
`kPDM_HwvadResultOREnergyBasedDetection`,  
`kPDM_HwvadResultANDEnergyBasedDetection` }  
*PDM voice activity detector zero cross detector result.*

## Driver version

- #define `FSL_PDM_DRIVER_VERSION` (`MAKE_VERSION`(2, 8, 0))  
*Version 2.8.0.*

## Initialization and deinitialization

- void `PDM_Init` (PDM\_Type \*base, const `pdm_config_t` \*config)  
*Initializes the PDM peripheral.*
- void `PDM_Deinit` (PDM\_Type \*base)  
*De-initializes the PDM peripheral.*
- static void `PDM_Reset` (PDM\_Type \*base)  
*Resets the PDM module.*
- static void `PDM_Enable` (PDM\_Type \*base, bool enable)  
*Enables/disables PDM interface.*
- static void `PDM_EnableDoze` (PDM\_Type \*base, bool enable)  
*Enables/disables DOZE.*
- static void `PDM_EnableDebugMode` (PDM\_Type \*base, bool enable)  
*Enables/disables debug mode for PDM.*
- static void `PDM_EnableInDebugMode` (PDM\_Type \*base, bool enable)  
*Enables/disables PDM interface in debug mode.*
- static void `PDM_EnterLowLeakageMode` (PDM\_Type \*base, bool enable)  
*Enables/disables PDM interface disable/Low Leakage mode.*
- static void `PDM_EnableChannel` (PDM\_Type \*base, uint8\_t channel, bool enable)  
*Enables/disables the PDM channel.*
- void `PDM_SetChannelConfig` (PDM\_Type \*base, uint32\_t channel, const `pdm_channel_config_t` \*config)  
*PDM one channel configurations.*
- `status_t` `PDM_SetSampleRateConfig` (PDM\_Type \*base, uint32\_t sourceClock\_HZ, uint32\_t sampleRate\_HZ)  
*PDM set sample rate.*
- `status_t` `PDM_SetSampleRate` (PDM\_Type \*base, uint32\_t enableChannelMask, `pdm_df_quality_mode_t` qualityMode, uint8\_t osr, uint32\_t clkDiv)  
*PDM set sample rate.*
- uint32\_t `PDM_GetInstance` (PDM\_Type \*base)  
*Get the instance number for PDM.*

## Status

- static uint32\_t [PDM\\_GetStatus](#) (PDM\_Type \*base)  
*Gets the PDM internal status flag.*
- static uint32\_t [PDM\\_GetFifoStatus](#) (PDM\_Type \*base)  
*Gets the PDM FIFO status flag.*
- static uint32\_t [PDM\\_GetOutputStatus](#) (PDM\_Type \*base)  
*Gets the PDM output status flag.*
- static void [PDM\\_ClearStatus](#) (PDM\_Type \*base, uint32\_t mask)  
*Clears the PDM Tx status.*
- static void [PDM\\_ClearFIFOStatus](#) (PDM\_Type \*base, uint32\_t mask)  
*Clears the PDM Tx status.*
- static void [PDM\\_ClearOutputStatus](#) (PDM\_Type \*base, uint32\_t mask)  
*Clears the PDM output status.*

## Interrupts

- void [PDM\\_EnableInterrupts](#) (PDM\_Type \*base, uint32\_t mask)  
*Enables the PDM interrupt requests.*
- static void [PDM\\_DisableInterrupts](#) (PDM\_Type \*base, uint32\_t mask)  
*Disables the PDM interrupt requests.*

## DMA Control

- static void [PDM\\_EnableDMA](#) (PDM\_Type \*base, bool enable)  
*Enables/disables the PDM DMA requests.*
- static uint32\_t [PDM\\_GetDataRegisterAddress](#) (PDM\_Type \*base, uint32\_t channel)  
*Gets the PDM data register address.*

## Bus Operations

- void [PDM\\_ReadFifo](#) (PDM\_Type \*base, uint32\_t startChannel, uint32\_t channelNums, void \*buffer, size\_t size, uint32\_t dataWidth)  
*PDM read fifo.*
- void [PDM\\_SetChannelGain](#) (PDM\_Type \*base, uint32\_t channel, [pdm\\_df\\_output\\_gain\\_t](#) gain)  
*Set the PDM channel gain.*

## Voice Activity Detector

- void [PDM\\_SetHwvadConfig](#) (PDM\_Type \*base, const [pdm\\_hwvad\\_config\\_t](#) \*config)  
*Configure voice activity detector.*
- static void [PDM\\_ForceHwvadOutputDisable](#) (PDM\_Type \*base, bool enable)  
*PDM hwvad force output disable.*
- static void [PDM\\_ResetHwvad](#) (PDM\_Type \*base)  
*PDM hwvad reset.*

- static void [PDM\\_EnableHwvad](#) (PDM\_Type \*base, bool enable)  
*Enable/Disable Voice activity detector.*
- static void [PDM\\_EnableHwvadInterrupts](#) (PDM\_Type \*base, uint32\_t mask)  
*Enables the PDM Voice Detector interrupt requests.*
- static void [PDM\\_DisableHwvadInterrupts](#) (PDM\_Type \*base, uint32\_t mask)  
*Disables the PDM Voice Detector interrupt requests.*
- static void [PDM\\_ClearHwvadInterruptStatusFlags](#) (PDM\_Type \*base, uint32\_t mask)  
*Clears the PDM voice activity detector status flags.*
- static uint32\_t [PDM\\_GetHwvadInterruptStatusFlags](#) (PDM\_Type \*base)  
*Clears the PDM voice activity detector status flags.*
- static uint32\_t [PDM\\_GetHwvadInitialFlag](#) (PDM\_Type \*base)  
*Get the PDM voice activity detector initial flags.*
- static uint32\_t [PDM\\_GetHwvadVoiceDetectedFlag](#) (PDM\_Type \*base)  
*Get the PDM voice activity detector voice detected flags.*
- static void [PDM\\_EnableHwvadSignalFilter](#) (PDM\_Type \*base, bool enable)  
*Enables/disables voice activity detector signal filter.*
- void [PDM\\_SetHwvadSignalFilterConfig](#) (PDM\_Type \*base, bool enableMaxBlock, uint32\_t signalGain)  
*Configure voice activity detector signal filter.*
- void [PDM\\_SetHwvadNoiseFilterConfig](#) (PDM\_Type \*base, const [pdm\\_hwvad\\_noise\\_filter\\_t](#) \*config)  
*Configure voice activity detector noise filter.*
- static void [PDM\\_EnableHwvadZeroCrossDetector](#) (PDM\_Type \*base, bool enable)  
*Enables/disables voice activity detector zero cross detector.*
- void [PDM\\_SetHwvadZeroCrossDetectorConfig](#) (PDM\_Type \*base, const [pdm\\_hwvad\\_zero\\_cross\\_detector\\_t](#) \*config)  
*Configure voice activity detector zero cross detector.*
- static uint16\_t [PDM\\_GetNoiseData](#) (PDM\_Type \*base)  
*Reads noise data.*
- static void [PDM\\_SetHwvadInternalFilterStatus](#) (PDM\_Type \*base, [pdm\\_hwvad\\_filter\\_status\\_t](#) status)  
*set hwvad internal filter status .*
- void [PDM\\_SetHwvadInEnvelopeBasedMode](#) (PDM\_Type \*base, const [pdm\\_hwvad\\_config\\_t](#) \*hwvadConfig, const [pdm\\_hwvad\\_noise\\_filter\\_t](#) \*noiseConfig, const [pdm\\_hwvad\\_zero\\_cross\\_detector\\_t](#) \*zcdConfig, uint32\_t signalGain)  
*set HWVAD in envelope based mode .*
- void [PDM\\_SetHwvadInEnergyBasedMode](#) (PDM\_Type \*base, const [pdm\\_hwvad\\_config\\_t](#) \*hwvadConfig, const [pdm\\_hwvad\\_noise\\_filter\\_t](#) \*noiseConfig, const [pdm\\_hwvad\\_zero\\_cross\\_detector\\_t](#) \*zcdConfig, uint32\_t signalGain)  
*brief set HWVAD in energy based mode .*
- void [PDM\\_EnableHwvadInterruptCallback](#) (PDM\_Type \*base, [pdm\\_hwvad\\_callback\\_t](#) vadCallback, void \*userData, bool enable)  
*Enable/Disable hwvad callback.*

## Transactional

- void [PDM\\_TransferCreateHandle](#) (PDM\_Type \*base, [pdm\\_handle\\_t](#) \*handle, [pdm\\_transfer\\_callback\\_t](#) callback, void \*userData)

- *Initializes the PDM handle.*
- `status_t PDM_TransferSetChannelConfig` (PDM\_Type \*base, pdm\_handle\_t \*handle, uint32\_t channel, const `pdm_channel_config_t` \*config, uint32\_t format)
- *PDM set channel transfer config.*
- `status_t PDM_TransferReceiveNonBlocking` (PDM\_Type \*base, pdm\_handle\_t \*handle, `pdm_transfer_t` \*xfer)
- *Performs an interrupt non-blocking receive transfer on PDM.*
- `void PDM_TransferAbortReceive` (PDM\_Type \*base, pdm\_handle\_t \*handle)
- *Aborts the current IRQ receive.*
- `void PDM_TransferHandleIRQ` (PDM\_Type \*base, pdm\_handle\_t \*handle)
- *Tx interrupt handler.*

### 16.3.3 Data Structure Documentation

#### 16.3.3.1 struct pdm\_channel\_config\_t

##### Data Fields

- `pdm_dc_removal_t cutOffFreq`  
*DC remover cut off frequency.*
- `pdm_df_output_gain_t gain`  
*Decimation Filter Output Gain.*

#### 16.3.3.2 struct pdm\_config\_t

##### Data Fields

- `bool enableDoze`  
*This module will enter disable/low leakage mode if DOZEN is active with ipg\_doze is asserted.*
- `uint8_t fifoWatermark`  
*Watermark value for FIFO.*
- `pdm_df_quality_mode_t qualityMode`  
*Quality mode.*
- `uint8_t cicOverSampleRate`  
*CIC filter over sampling rate.*

#### 16.3.3.3 struct pdm\_hwvad\_config\_t

##### Data Fields

- `uint8_t channel`  
*Which channel uses voice activity detector.*
- `uint8_t initializeTime`  
*Number of frames or samples to initialize voice activity detector.*
- `uint8_t cicOverSampleRate`  
*CIC filter over sampling rate.*
- `uint8_t inputGain`

- `uint32_t frameTime`  
*Voice activity detector input gain.*
- `pdm_hwvad_hpf_config_t cutOffFreq`  
*Voice activity frame time.*
- `bool enableFrameEnergy`  
*High pass filter cut off frequency.*
- `bool enablePreFilter`  
*If frame energy enabled, true means enable.*
- `bool enablePreFilter`  
*If pre-filter enabled.*

## Field Documentation

### (1) `uint8_t pdm_hwvad_config_t::initializeTime`

#### 16.3.3.4 `struct pdm_hwvad_noise_filter_t`

##### Data Fields

- `bool enableAutoNoiseFilter`  
*If noise fileter automatically activated, true means enable.*
- `bool enableNoiseMin`  
*If Noise minimum block enabled, true means enabled.*
- `bool enableNoiseDecimation`  
*If enable noise input decimation.*
- `bool enableNoiseDetectOR`  
*Enables a OR logic in the output of minimum noise estimator block.*
- `uint32_t noiseFilterAdjustment`  
*The adjustment value of the noise filter.*
- `uint32_t noiseGain`  
*Gain value for the noise energy or envelope estimated.*

#### 16.3.3.5 `struct pdm_hwvad_zero_cross_detector_t`

##### Data Fields

- `bool enableAutoThreshold`  
*If ZCD auto-threshold enabled, true means enabled.*
- `pdm_hwvad_zcd_result_t zcdAnd`  
*Is ZCD result is AND'ed with energy-based detection, false means OR'ed.*
- `uint32_t threshold`  
*The adjustment value of the noise filter.*
- `uint32_t adjustmentThreshold`  
*Gain value for the noise energy or envelope estimated.*

## Field Documentation

(1) `bool pdm_hwvad_zero_cross_detector_t::enableAutoThreshold`

### 16.3.3.6 struct `pdm_transfer_t`

#### Data Fields

- `volatile uint8_t * data`  
*Data start address to transfer.*
- `volatile size_t dataSize`  
*Total Transfer bytes size.*

## Field Documentation

(1) `volatile uint8_t* pdm_transfer_t::data`

(2) `volatile size_t pdm_transfer_t::dataSize`

### 16.3.3.7 struct `pdm_hwvad_notification_t`

### 16.3.3.8 struct `_pdm_handle`

PDM handle.

#### Data Fields

- `uint32_t state`  
*Transfer status.*
- `pdm_transfer_callback_t callback`  
*Callback function called at transfer event.*
- `void * userData`  
*Callback parameter passed to callback function.*
- `pdm_transfer_t pdmQueue [PDM_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSize [PDM_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.*
- `uint32_t format`  
*data format*
- `uint8_t watermark`  
*Watermark value.*
- `uint8_t startChannel`  
*end channel*
- `uint8_t channelNums`  
*Enabled channel number.*

### 16.3.4 Enumeration Type Documentation

#### 16.3.4.1 anonymous enum

Enumerator

*kStatus\_PDM\_Busy* PDM is busy.  
*kStatus\_PDM\_FIFO\_ERROR* PDM FIFO underrun or overflow.  
*kStatus\_PDM\_QueueFull* PDM FIFO underrun or overflow.  
*kStatus\_PDM\_Idle* PDM is idle.  
*kStatus\_PDM\_Output\_ERROR* PDM is output error.  
*kStatus\_PDM\_ChannelConfig\_Failed* PDM channel config failed.  
*kStatus\_PDM\_HWVAD\_VoiceDetected* PDM hwvad voice detected.  
*kStatus\_PDM\_HWVAD\_Error* PDM hwvad error.

#### 16.3.4.2 enum \_pdm\_interrupt\_enable

Enumerator

*kPDM\_ErrorInterruptEnable* PDM channel error interrupt enable.  
*kPDM\_FIFOInterruptEnable* PDM channel FIFO interrupt.

#### 16.3.4.3 enum \_pdm\_internal\_status

Enumerator

*kPDM\_StatusDfBusyFlag* Decimation filter is busy processing data.  
*kPDM\_StatusFIRFilterReady* FIR filter data is ready.  
*kPDM\_StatusCh0FifoDataAvaliable* channel 0 fifo data reached watermark level  
*kPDM\_StatusCh1FifoDataAvaliable* channel 1 fifo data reached watermark level  
*kPDM\_StatusCh2FifoDataAvaliable* channel 2 fifo data reached watermark level  
*kPDM\_StatusCh3FifoDataAvaliable* channel 3 fifo data reached watermark level  
*kPDM\_StatusCh4FifoDataAvaliable* channel 4 fifo data reached watermark level  
*kPDM\_StatusCh5FifoDataAvaliable* channel 5 fifo data reached watermark level  
*kPDM\_StatusCh6FifoDataAvaliable* channel 6 fifo data reached watermark level  
*kPDM\_StatusCh7FifoDataAvaliable* channel 7 fifo data reached watermark level

#### 16.3.4.4 enum \_pdm\_channel\_enable\_mask

Enumerator

*kPDM\_EnableChannel0* channel 0 enable mask  
*kPDM\_EnableChannel1* channel 1 enable mask



***kPDM\_EnableChannel2***    channel 2 enable mask  
***kPDM\_EnableChannel3***    channel 3 enable mask  
***kPDM\_EnableChannel4***    channel 4 enable mask  
***kPDM\_EnableChannel5***    channel 5 enable mask  
***kPDM\_EnableChannel6***    channel 6 enable mask  
***kPDM\_EnableChannel7***    channel 7 enable mask

#### 16.3.4.5 enum \_pdm\_fifo\_status

Enumerator

***kPDM\_FifoStatusUnderflowCh0***    channel0 fifo status underflow  
***kPDM\_FifoStatusUnderflowCh1***    channel1 fifo status underflow  
***kPDM\_FifoStatusUnderflowCh2***    channel2 fifo status underflow  
***kPDM\_FifoStatusUnderflowCh3***    channel3 fifo status underflow  
***kPDM\_FifoStatusUnderflowCh4***    channel4 fifo status underflow  
***kPDM\_FifoStatusUnderflowCh5***    channel5 fifo status underflow  
***kPDM\_FifoStatusUnderflowCh6***    channel6 fifo status underflow  
***kPDM\_FifoStatusUnderflowCh7***    channel7 fifo status underflow  
***kPDM\_FifoStatusOverflowCh0***    channel0 fifo status overflow  
***kPDM\_FifoStatusOverflowCh1***    channel1 fifo status overflow  
***kPDM\_FifoStatusOverflowCh2***    channel2 fifo status overflow  
***kPDM\_FifoStatusOverflowCh3***    channel3 fifo status overflow  
***kPDM\_FifoStatusOverflowCh4***    channel4 fifo status overflow  
***kPDM\_FifoStatusOverflowCh5***    channel5 fifo status overflow  
***kPDM\_FifoStatusOverflowCh6***    channel6 fifo status overflow  
***kPDM\_FifoStatusOverflowCh7***    channel7 fifo status overflow

#### 16.3.4.6 enum \_pdm\_output\_status

Enumerator

***kPDM\_OutputStatusUnderFlowCh0***    channel0 output status underflow  
***kPDM\_OutputStatusUnderFlowCh1***    channel1 output status underflow  
***kPDM\_OutputStatusUnderFlowCh2***    channel2 output status underflow  
***kPDM\_OutputStatusUnderFlowCh3***    channel3 output status underflow  
***kPDM\_OutputStatusUnderFlowCh4***    channel4 output status underflow  
***kPDM\_OutputStatusUnderFlowCh5***    channel5 output status underflow  
***kPDM\_OutputStatusUnderFlowCh6***    channel6 output status underflow  
***kPDM\_OutputStatusUnderFlowCh7***    channel7 output status underflow  
***kPDM\_OutputStatusOverFlowCh0***    channel0 output status overflow  
***kPDM\_OutputStatusOverFlowCh1***    channel1 output status overflow  
***kPDM\_OutputStatusOverFlowCh2***    channel2 output status overflow

***kPDM\_OutputStatusOverflowCh3*** channel3 output status overflow  
***kPDM\_OutputStatusOverflowCh4*** channel4 output status overflow  
***kPDM\_OutputStatusOverflowCh5*** channel5 output status overflow  
***kPDM\_OutputStatusOverflowCh6*** channel6 output status overflow  
***kPDM\_OutputStatusOverflowCh7*** channel7 output status overflow

#### 16.3.4.7 enum pdm\_dc\_removal\_t

Enumerator

***kPDM\_DcRemoverCutOff21Hz*** DC remover cut off 21HZ.  
***kPDM\_DcRemoverCutOff83Hz*** DC remover cut off 83HZ.  
***kPDM\_DcRemoverCutOff152Hz*** DC remover cut off 152HZ.  
***kPDM\_DcRemoverBypass*** DC remover bypass.

#### 16.3.4.8 enum pdm\_df\_quality\_mode\_t

Enumerator

***kPDM\_QualityModeMedium*** quality mode medium  
***kPDM\_QualityModeHigh*** quality mode high  
***kPDM\_QualityModeLow*** quality mode low  
***kPDM\_QualityModeVeryLow0*** quality mode very low0  
***kPDM\_QualityModeVeryLow1*** quality mode very low1  
***kPDM\_QualityModeVeryLow2*** quality mode very low2

#### 16.3.4.9 enum \_pdm\_quality\_mode\_k\_factor

Enumerator

***kPDM\_QualityModeHighKFactor*** high quality mode K factor = 1 / 2  
***kPDM\_QualityModeMediumKFactor*** medium/very low0 quality mode K factor = 2 / 2  
***kPDM\_QualityModeLowKFactor*** low/very low1 quality mode K factor = 4 / 2  
***kPDM\_QualityModeVeryLow2KFactor*** very low2 quality mode K factor = 8 / 2

#### 16.3.4.10 enum pdm\_df\_output\_gain\_t

Enumerator

***kPDM\_DfOutputGain0*** Decimation filter output gain 0.  
***kPDM\_DfOutputGain1*** Decimation filter output gain 1.  
***kPDM\_DfOutputGain2*** Decimation filter output gain 2.

***kPDM\_DfOutputGain3*** Decimation filter output gain 3.  
***kPDM\_DfOutputGain4*** Decimation filter output gain 4.  
***kPDM\_DfOutputGain5*** Decimation filter output gain 5.  
***kPDM\_DfOutputGain6*** Decimation filter output gain 6.  
***kPDM\_DfOutputGain7*** Decimation filter output gain 7.  
***kPDM\_DfOutputGain8*** Decimation filter output gain 8.  
***kPDM\_DfOutputGain9*** Decimation filter output gain 9.  
***kPDM\_DfOutputGain10*** Decimation filter output gain 10.  
***kPDM\_DfOutputGain11*** Decimation filter output gain 11.  
***kPDM\_DfOutputGain12*** Decimation filter output gain 12.  
***kPDM\_DfOutputGain13*** Decimation filter output gain 13.  
***kPDM\_DfOutputGain14*** Decimation filter output gain 14.  
***kPDM\_DfOutputGain15*** Decimation filter output gain 15.

#### 16.3.4.11 enum \_pdm\_data\_width

Enumerator

***kPDM\_DataWidth16*** PDM data width 16bit.

#### 16.3.4.12 enum \_pdm\_hwvad\_interrupt\_enable

Enumerator

***kPDM\_HwvadErrorInterruptEnable*** PDM channel HWVAD error interrupt enable.  
***kPDM\_HwvadInterruptEnable*** PDM channel HWVAD interrupt.

#### 16.3.4.13 enum \_pdm\_hwvad\_int\_status

Enumerator

***kPDM\_HwvadStatusInputSaturation*** HWVAD saturation condition.  
***kPDM\_HwvadStatusVoiceDetectFlag*** HWVAD voice detect interrupt triggered.

#### 16.3.4.14 enum pdm\_hwvad\_hpf\_config\_t

Enumerator

***kPDM\_HwvadHpfBypassed*** High-pass filter bypass.  
***kPDM\_HwvadHpfCutOffFreq1750Hz*** High-pass filter cut off frequency 1750HZ.  
***kPDM\_HwvadHpfCutOffFreq215Hz*** High-pass filter cut off frequency 215HZ.  
***kPDM\_HwvadHpfCutOffFreq102Hz*** High-pass filter cut off frequency 102HZ.

#### 16.3.4.15 enum pdm\_hwvad\_filter\_status\_t

Enumerator

***kPDM\_HwvadInternalFilterNormalOperation*** internal filter ready for normal operation

***kPDM\_HwvadInternalFilterInitial*** interla filter are initial

#### 16.3.4.16 enum pdm\_hwvad\_zcd\_result\_t

Enumerator

***kPDM\_HwvadResultOREnergyBasedDetection*** zero cross detector result will be OR with energy based detection

***kPDM\_HwvadResultANDEnergyBasedDetection*** zero cross detector result will be AND with energy based detection

### 16.3.5 Function Documentation

#### 16.3.5.1 void PDM\_Init ( PDM\_Type \* *base*, const pdm\_config\_t \* *config* )

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by PDM\_GetDefaultConfig().

Note

This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PDM base pointer             |
| <i>config</i> | PDM configuration structure. |

#### 16.3.5.2 void PDM\_Deinit ( PDM\_Type \* *base* )

This API gates the PDM clock. The PDM module can't operate unless PDM\_Init is called to enable the clock.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

### 16.3.5.3 static void PDM\_Reset ( PDM\_Type \* *base* ) [inline], [static]

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

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

Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                            |
| <i>enable</i> | True means PDM interface is enabled, false means PDM interface is disabled. |

### 16.3.5.5 static void PDM\_EnableDoze ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                                                                                                                                |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                                                                                                                               |
| <i>enable</i> | True means the module will enter Disable/Low Leakage mode when ipg_doze is asserted, false means the module will not enter Disable/Low Leakage mode when ipg_doze is asserted. |

### 16.3.5.6 static void PDM\_EnableDebugMode ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]

The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

Parameters

---

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                                     |
| <i>enable</i> | True means PDM interface enter debug mode, false means PDM interface in normal mode. |

**16.3.5.7 static void PDM\_EnableInDebugMode ( PDM\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

Parameters

|               |                                                                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                                                                                              |
| <i>enable</i> | True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode. |

**16.3.5.8 static void PDM\_EnterLowLeakageMode ( PDM\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

Parameters

|               |                                                                                                       |
|---------------|-------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                                                      |
| <i>enable</i> | True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode. |

**16.3.5.9 static void PDM\_EnableChannel ( PDM\_Type \* *base*, uint8\_t *channel*, bool *enable* )**  
**[inline], [static]**

Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | PDM base pointer                                    |
| <i>channel</i> | PDM channel number need to enable or disable.       |
| <i>enable</i>  | True means enable PDM channel, false means disable. |

**16.3.5.10 void PDM\_SetChannelConfig ( PDM\_Type \* *base*, uint32\_t *channel*, const pdm\_channel\_config\_t \* *config* )**

## Parameters

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| <i>base</i>    | PDM base pointer                                                  |
| <i>config</i>  | PDM channel configurations.                                       |
| <i>channel</i> | channel number. after completing the current frame in debug mode. |

### 16.3.5.11 **status\_t PDM\_SetSampleRateConfig ( PDM\_Type \* *base*, uint32\_t *sourceClock\_HZ*, uint32\_t *sampleRate\_HZ* )**

## Note

This function is depend on the configuration of the PDM and PDM channel, so the correct call sequence is

```
* PDM_Init(base, pdmConfig)
* PDM_SetChannelConfig(base, channel, &channelConfig)
* PDM_SetSampleRateConfig(base, source, sampleRate)
*
```

## Parameters

|                       |                             |
|-----------------------|-----------------------------|
| <i>base</i>           | PDM base pointer            |
| <i>sourceClock_HZ</i> | PDM source clock frequency. |
| <i>sampleRate_HZ</i>  | PDM sample rate.            |

### 16.3.5.12 **status\_t PDM\_SetSampleRate ( PDM\_Type \* *base*, uint32\_t *enableChannelMask*, pdm\_df\_quality\_mode\_t *qualityMode*, uint8\_t *osr*, uint32\_t *clkDiv* )**

**Deprecated** Do not use this function. It has been supersceded by [PDM\\_SetSampleRateConfig](#)

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

|                           |                          |
|---------------------------|--------------------------|
| <i>enable-ChannelMask</i> | PDM channel enable mask. |
| <i>qualityMode</i>        | quality mode.            |
| <i>osr</i>                | cic oversample rate      |
| <i>clkDiv</i>             | clock divider            |

#### 16.3.5.13 uint32\_t PDM\_GetInstance ( PDM\_Type \* *base* )

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | PDM base pointer. |
|-------------|-------------------|

#### 16.3.5.14 static uint32\_t PDM\_GetStatus ( PDM\_Type \* *base* ) [inline], [static]

Use the Status Mask in `_pdm_internal_status` to get the status value needed

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

Returns

PDM status flag value.

#### 16.3.5.15 static uint32\_t PDM\_GetFifoStatus ( PDM\_Type \* *base* ) [inline], [static]

Use the Status Mask in `_pdm_fifo_status` to get the status value needed

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

Returns

FIFO status.

#### 16.3.5.16 static uint32\_t PDM\_GetOutputStatus ( PDM\_Type \* *base* ) [inline], [static]

Use the Status Mask in `_pdm_output_status` to get the status value needed



## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

## Returns

output status.

**16.3.5.17 static void PDM\_ClearStatus ( PDM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM base pointer                                                                                                        |
| <i>mask</i> | State mask. It can be a combination of the status between kPDM_StatusFrequency-Low and kPDM_StatusCh7FifoDataAvaliable. |

**16.3.5.18 static void PDM\_ClearFIFOStatus ( PDM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |                                                                        |
|-------------|------------------------------------------------------------------------|
| <i>base</i> | PDM base pointer                                                       |
| <i>mask</i> | State mask. It can be a combination of the status in _pdm_fifo_status. |

**16.3.5.19 static void PDM\_ClearOutputStatus ( PDM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |                                                                          |
|-------------|--------------------------------------------------------------------------|
| <i>base</i> | PDM base pointer                                                         |
| <i>mask</i> | State mask. It can be a combination of the status in _pdm_output_status. |

**16.3.5.20 void PDM\_EnableInterrupts ( PDM\_Type \* *base*, uint32\_t *mask* )**

## Parameters

|             |                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM 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>• kPDM_ErrorInterruptEnable</li> <li>• kPDM_FIFOInterruptEnable</li> </ul> |

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

## Parameters

|             |                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM 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>• kPDM_ErrorInterruptEnable</li> <li>• kPDM_FIFOInterruptEnable</li> </ul> |

### 16.3.5.22 static void PDM\_EnableDMA ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | PDM base pointer                                |
| <i>enable</i> | True means enable DMA, false means disable DMA. |

### 16.3.5.23 static uint32\_t PDM\_GetDataRegisterAddress ( PDM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This API is used to provide a transfer address for the PDM DMA transfer configuration.

## Parameters

|                |                    |
|----------------|--------------------|
| <i>base</i>    | PDM base pointer   |
| <i>channel</i> | PDM channel number |

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | PDM base pointer.        |
| <i>channel</i> | Which data channel used. |

Returns

data register address.

**16.3.5.24 void PDM\_ReadFifo ( PDM\_Type \* *base*, uint32\_t *startChannel*, uint32\_t *channelNums*, void \* *buffer*, size\_t *size*, uint32\_t *dataWidth* )**

Note

: This function support 16 bit only for IP version that only supports 16bit.

Parameters

|                     |                            |
|---------------------|----------------------------|
| <i>base</i>         | PDM base pointer.          |
| <i>startChannel</i> | start channel number.      |
| <i>channelNums</i>  | total enabled channelnums. |
| <i>buffer</i>       | received buffer address.   |
| <i>size</i>         | number of samples to read. |
| <i>dataWidth</i>    | sample width.              |

**16.3.5.25 void PDM\_SetChannelGain ( PDM\_Type \* *base*, uint32\_t *channel*, pdm\_df\_output\_gain\_t *gain* )**

Please note for different quality mode, the valid gain value is different, reference RM for detail.

Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>base</i>    | PDM base pointer.                                      |
| <i>channel</i> | PDM channel index.                                     |
| <i>gain</i>    | channel gain, the register gain value range is 0 - 15. |

**16.3.5.26 void PDM\_SetHwvadConfig ( PDM\_Type \* *base*, const pdm\_hwvad\_config\_t \* *config* )**

## Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                      |
| <i>config</i> | Voice activity detector configure structure pointer . |

**16.3.5.27 static void PDM\_ForceHwvadOutputDisable ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                         |
| <i>enable</i> | true is output force disable, false is output not force. |

**16.3.5.28 static void PDM\_ResetHwvad ( PDM\_Type \* *base* ) [inline], [static]**

It will reset VADNDATA register and will clean all internal buffers, should be called when the PDM isn't running.

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

**16.3.5.29 static void PDM\_EnableHwvad ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]**

Should be called when the PDM isn't running.

## Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>base</i>   | PDM base pointer.                                               |
| <i>enable</i> | True means enable voice activity detector, false means disable. |

**16.3.5.30 static void PDM\_EnableHwvadInterrupts ( PDM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |                                                                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM 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>• kPDM_HWVADErrorInterruptEnable</li> <li>• kPDM_HWVADInterruptEnable</li> </ul> |

**16.3.5.31** `static void PDM_DisableHwvadInterrupts ( PDM_Type * base, uint32_t mask )`  
**[inline], [static]**

## Parameters

|             |                                                                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM 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>• kPDM_HWVADErrorInterruptEnable</li> <li>• kPDM_HWVADInterruptEnable</li> </ul> |

**16.3.5.32** `static void PDM_ClearHwvadInterruptStatusFlags ( PDM_Type * base, uint32_t mask )`  
**[inline], [static]**

## Parameters

|             |                                             |
|-------------|---------------------------------------------|
| <i>base</i> | PDM base pointer                            |
| <i>mask</i> | State mask,reference _pdm_hwvad_int_status. |

**16.3.5.33** `static uint32_t PDM_GetHwvadInterruptStatusFlags ( PDM_Type * base )`  
**[inline], [static]**

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

Returns

status, reference `_pdm_hwvad_int_status`

**16.3.5.34** `static uint32_t PDM_GetHwvadInitialFlag ( PDM_Type * base ) [inline], [static]`

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

Returns

initial flag.

**16.3.5.35** `static uint32_t PDM_GetHwvadVoiceDetectedFlag ( PDM_Type * base ) [inline], [static]`

NOte: this flag is auto cleared when voice gone.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PDM base pointer |
|-------------|------------------|

Returns

voice detected flag.

**16.3.5.36** `static void PDM_EnableHwvadSignalFilter ( PDM_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                      |
| <i>enable</i> | True means enable signal filter, false means disable. |

**16.3.5.37 void PDM\_SetHwvadSignalFilterConfig ( PDM\_Type \* *base*, bool *enableMaxBlock*, uint32\_t *signalGain* )**

Parameters

|                        |                                   |
|------------------------|-----------------------------------|
| <i>base</i>            | PDM base pointer                  |
| <i>enableMax-Block</i> | If signal maximum block enabled.  |
| <i>signalGain</i>      | Gain value for the signal energy. |

**16.3.5.38 void PDM\_SetHwvadNoiseFilterConfig ( PDM\_Type \* *base*, const pdm\_hwvad\_noise\_filter\_t \* *config* )**

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                   |
| <i>config</i> | Voice activity detector noise filter configure structure pointer . |

**16.3.5.39 static void PDM\_EnableHwvadZeroCrossDetector ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                            |
| <i>enable</i> | True means enable zero cross detector, false means disable. |

**16.3.5.40 void PDM\_SetHwvadZeroCrossDetectorConfig ( PDM\_Type \* *base*, const pdm\_hwvad\_zero\_cross\_detector\_t \* *config* )**

## Parameters

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                          |
| <i>config</i> | Voice activity detector zero cross detector configure structure pointer . |

**16.3.5.41** `static uint16_t PDM_GetNoiseData ( PDM_Type * base ) [inline], [static]`

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | PDM base pointer. |
|-------------|-------------------|

## Returns

Data in PDM noise data register.

**16.3.5.42** `static void PDM_SetHwvadInternalFilterStatus ( PDM_Type * base,  
pdm_hwvad_filter_status_t status ) [inline], [static]`

Note: filter initial status should be asserted for two more cycles, then set it to normal operation.

## Parameters

|               |                         |
|---------------|-------------------------|
| <i>base</i>   | PDM base pointer.       |
| <i>status</i> | internal filter status. |

**16.3.5.43** `void PDM_SetHwvadInEnvelopeBasedMode ( PDM_Type * base, const  
pdm_hwvad_config_t * hwvadConfig, const pdm_hwvad_noise_filter_t *  
noiseConfig, const pdm_hwvad_zero_cross_detector_t * zcdConfig, uint32_t  
signalGain )`

## Recommand configurations,

```
* static const pdm_hwvad_config_t hwvadConfig = {
* .channel = 0,
* .initializeTime = 10U,
* .cicOverSampleRate = 0U,
* .inputGain = 0U,
* .frameTime = 10U,
* .cutOffFreq = kPDM_HwvadHpfBypassed,
* .enableFrameEnergy = false,
* .enablePreFilter = true,
* };
```



```

* static const pdm_hwvad_noise_filter_t noiseFilterConfig = {
* .enableAutoNoiseFilter = false,
* .enableNoiseMin = true,
* .enableNoiseDecimation = true,
* .noiseFilterAdjustment = 0U,
* .noiseGain = 7U,
* .enableNoiseDetectOR = true,
* };
*

```

#### Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>base</i>        | PDM base pointer.                                                         |
| <i>hwvadConfig</i> | internal filter status.                                                   |
| <i>noiseConfig</i> | Voice activity detector noise filter configure structure pointer.         |
| <i>zcdConfig</i>   | Voice activity detector zero cross detector configure structure pointer . |
| <i>signalGain</i>  | signal gain value.                                                        |

**16.3.5.44 void PDM\_SetHwvadInEnergyBasedMode ( PDM\_Type \* *base*, const pdm\_hwvad\_config\_t \* *hwvadConfig*, const pdm\_hwvad\_noise\_filter\_t \* *noiseConfig*, const pdm\_hwvad\_zero\_cross\_detector\_t \* *zcdConfig*, uint32\_t *signalGain* )**

Recommand configurations, code static const `pdm_hwvad_config_t` `hwvadConfig` = { .channel = 0, .initializeTime = 10U, .cicOverSampleRate = 0U, .inputGain = 0U, .frameTime = 10U, .cutOffFreq = kPDM\_HwvadHpfBypassed, .enableFrameEnergy = true, .enablePreFilter = true, };

static const `pdm_hwvad_noise_filter_t` `noiseFilterConfig` = { .enableAutoNoiseFilter = true, .enableNoiseMin = false, .enableNoiseDecimation = false, .noiseFilterAdjustment = 0U, .noiseGain = 7U, .enableNoiseDetectOR = false, }; code param *base* PDM base pointer. param *hwvadConfig* internal filter status. param *noiseConfig* Voice activity detector noise filter configure structure pointer. param *zcdConfig* Voice activity detector zero cross detector configure structure pointer . param *signalGain* signal gain value, signal gain value should be properly according to application.

**16.3.5.45 void PDM\_EnableHwvadInterruptCallback ( PDM\_Type \* *base*, pdm\_hwvad\_callback\_t *vadCallback*, void \* *userData*, bool *enable* )**

This function enable/disable the hwvad interrupt for the selected PDM peripheral.

#### Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>base</i>        | Base address of the PDM peripheral.                                       |
| <i>vadCallback</i> | callback Pointer to store callback function, should be NULL when disable. |
| <i>userData</i>    | user data.                                                                |
| <i>enable</i>      | true is enable, false is disable.                                         |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 16.3.5.46 void PDM\_TransferCreateHandle ( PDM\_Type \* *base*, pdm\_handle\_t \* *handle*, pdm\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the handle for the PDM transactional APIs. Call this function once to get the handle initialized.

Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>base</i>     | PDM base pointer.                               |
| <i>handle</i>   | PDM handle pointer.                             |
| <i>callback</i> | Pointer to the user callback function.          |
| <i>userData</i> | User parameter passed to the callback function. |

#### 16.3.5.47 status\_t PDM\_TransferSetChannelConfig ( PDM\_Type \* *base*, pdm\_handle\_t \* *handle*, uint32\_t *channel*, const pdm\_channel\_config\_t \* *config*, uint32\_t *format* )

Parameters

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| <i>base</i>    | PDM base pointer.                                                |
| <i>handle</i>  | PDM handle pointer.                                              |
| <i>channel</i> | PDM channel.                                                     |
| <i>config</i>  | channel config.                                                  |
| <i>format</i>  | data format, support data width configurations, _pdm_data_width. |

## Return values

|                                          |                     |
|------------------------------------------|---------------------|
| <i>kStatus_PDM_Channel-Config_Failed</i> | or kStatus_Success. |
|------------------------------------------|---------------------|

### 16.3.5.48 status\_t PDM\_TransferReceiveNonBlocking ( PDM\_Type \* *base*, pdm\_handle\_t \* *handle*, pdm\_transfer\_t \* *xfer* )

## Note

This API returns immediately after the transfer initiates. Call the PDM\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_PDM\_Busy, the transfer is finished.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                       |
| <i>handle</i> | Pointer to the pdm_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <a href="#">pdm_transfer_t</a> structure.               |

## Return values

|                         |                                        |
|-------------------------|----------------------------------------|
| <i>kStatus_Success</i>  | Successfully started the data receive. |
| <i>kStatus_PDM_Busy</i> | Previous receive still not finished.   |

### 16.3.5.49 void PDM\_TransferAbortReceive ( PDM\_Type \* *base*, pdm\_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>   | PDM base pointer                                                       |
| <i>handle</i> | Pointer to the pdm_handle_t structure which stores the transfer state. |

### 16.3.5.50 void PDM\_TransferHandleIRQ ( PDM\_Type \* *base*, pdm\_handle\_t \* *handle* )

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | PDM base pointer.                      |
| <i>handle</i> | Pointer to the pdm_handle_t structure. |

## 16.4 PDM SDMA Driver

### 16.4.1 Typical use case

### 16.4.2 Overview

The SDMA multi fifo script support transfer data between multi peripheral fifos and memory, a typical user case is that receiving multi PDM channel data and put it into memory as

| channel 0 | channel 1 | channel 2 | channel 3 | channel 4 | ..... |

Multi fifo script is target to implement above feature, it can supports 1.configurable fifo watermark range from  $1 \sim (2^{12} - 1)$ , it is a value of `fifo_watermark * channel_numbers` 2.configurable fifo numbers, support up to 15 continuous fifos 3.configurable fifo address offset, support address offset up to 64

```
/* load sdma script */
SDMA_LoadScript()
/* pdm multi channel configurations */
PDM_SetChannelConfigSDMA()
PDM_SetChannelConfigSDMA()
PDM_SetChannelConfigSDMA()
PDM_SetChannelConfigSDMA()
....
PDM_TransferReceiveSDMA
```

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

## Data Structures

- struct `pdm_sdma_handle_t`  
*PDM DMA transfer handle, users should not touch the content of the handle. [More...](#)*

## Typedefs

- typedef void(\* `pdm_sdma_callback_t`)(PDM\_Type \*base, pdm\_sdma\_handle\_t \*handle, `status_t` status, void \*userData)  
*PDM eDMA transfer callback function for finish and error.*

## Driver version

- #define `FSL_PDM_SDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 7, 0)`)  
*Version 2.7.0.*

## eDMA Transactional

- void [PDM\\_TransferCreateHandleSDMA](#) (PDM\_Type \*base, pdm\_sdma\_handle\_t \*handle, [pdm\\_sdma\\_callback\\_t](#) callback, void \*userData, [sdma\\_handle\\_t](#) \*dmaHandle, uint32\_t eventSource)  
*Initializes the PDM eDMA handle.*
- [status\\_t PDM\\_TransferReceiveSDMA](#) (PDM\_Type \*base, pdm\_sdma\_handle\_t \*handle, [pdm\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking PDM receive using eDMA.*
- void [PDM\\_TransferAbortReceiveSDMA](#) (PDM\_Type \*base, pdm\_sdma\_handle\_t \*handle)  
*Aborts a PDM receive using eDMA.*
- void [PDM\\_SetChannelConfigSDMA](#) (PDM\_Type \*base, pdm\_sdma\_handle\_t \*handle, uint32\_t channel, const [pdm\\_channel\\_config\\_t](#) \*config)  
*PDM channel configurations.*
- void [PDM\\_TransferTerminateReceiveSDMA](#) (PDM\_Type \*base, pdm\_sdma\_handle\_t \*handle)  
*Terminate all the PDM sdma receive transfer.*

## 16.4.3 Data Structure Documentation

### 16.4.3.1 struct \_pdm\_sdma\_handle

#### Data Fields

- [sdma\\_handle\\_t](#) \* [dmaHandle](#)  
*DMA handler for PDM send.*
- uint8\_t [nbytes](#)  
*eDMA minor byte transfer count initially configured.*
- uint8\_t [fifoWidth](#)  
*fifo width*
- uint8\_t [endChannel](#)  
*The last enabled channel.*
- uint8\_t [channelNums](#)  
*total channel numbers*
- uint32\_t [count](#)  
*The transfer data count in a DMA request.*
- uint32\_t [state](#)  
*Internal state for PDM eDMA transfer.*
- uint32\_t [eventSource](#)  
*PDM event source number.*
- [pdm\\_sdma\\_callback\\_t](#) [callback](#)  
*Callback for users while transfer finish or error occurs.*
- void \* [userData](#)  
*User callback parameter.*
- [sdma\\_buffer\\_descriptor\\_t](#) [bdPool](#) [[PDM\\_XFER\\_QUEUE\\_SIZE](#)]  
*BD pool for SDMA transfer.*
- [pdm\\_transfer\\_t](#) [pdmQueue](#) [[PDM\\_XFER\\_QUEUE\\_SIZE](#)]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [[PDM\\_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.*

## Field Documentation

- (1) uint8\_t `pdm_sdma_handle_t::nbytes`
- (2) sdma\_buffer\_descriptor\_t `pdm_sdma_handle_t::bdPool[PDM_XFER_QUEUE_SIZE]`
- (3) pdm\_transfer\_t `pdm_sdma_handle_t::pdmQueue[PDM_XFER_QUEUE_SIZE]`
- (4) volatile uint8\_t `pdm_sdma_handle_t::queueUser`

## 16.4.4 Function Documentation

**16.4.4.1 void PDM\_TransferCreateHandleSDMA ( PDM\_Type \* *base*, pdm\_sdma\_handle\_t \* *handle*, pdm\_sdma\_callback\_t *callback*, void \* *userData*, sdma\_handle\_t \* *dmaHandle*, uint32\_t *eventSource* )**

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

Parameters

|                    |                                                                      |
|--------------------|----------------------------------------------------------------------|
| <i>base</i>        | PDM base pointer.                                                    |
| <i>handle</i>      | PDM eDMA handle pointer.                                             |
| <i>callback</i>    | Pointer to user callback function.                                   |
| <i>userData</i>    | User parameter passed to the callback function.                      |
| <i>dmaHandle</i>   | eDMA handle pointer, this handle shall be static allocated by users. |
| <i>eventSource</i> | PDM event source number.                                             |

**16.4.4.2 status\_t PDM\_TransferReceiveSDMA ( PDM\_Type \* *base*, pdm\_sdma\_handle\_t \* *handle*, pdm\_transfer\_t \* *xfer* )**

Note

This interface returns immediately after the transfer initiates. Call the PDM\_GetReceiveRemaining-Bytes to poll the transfer status and check whether the PDM transfer is finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | PDM base pointer                   |
| <i>handle</i> | PDM eDMA handle pointer.           |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Start a PDM eDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.         |
| <i>kStatus_RxBusy</i>          | PDM is busy receiving data.            |

#### 16.4.4.3 void PDM\_TransferAbortReceiveSDMA ( PDM\_Type \* *base*, pdm\_sdma\_handle\_t \* *handle* )

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | PDM base pointer         |
| <i>handle</i> | PDM eDMA handle pointer. |

#### 16.4.4.4 void PDM\_SetChannelConfigSDMA ( PDM\_Type \* *base*, pdm\_sdma\_handle\_t \* *handle*, uint32\_t *channel*, const pdm\_channel\_config\_t \* *config* )

## Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | PDM base pointer.        |
| <i>handle</i>  | PDM eDMA handle pointer. |
| <i>channel</i> | channel number.          |
| <i>config</i>  | channel configurations.  |

#### 16.4.4.5 void PDM\_TransferTerminateReceiveSDMA ( PDM\_Type \* *base*, pdm\_sdma\_handle\_t \* *handle* )



## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | PDM base pointer.        |
| <i>handle</i> | PDM SDMA handle pointer. |

## Chapter 17

# RDC: Resource Domain Controller

### 17.1 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...](#)*

### Enumerations

- enum [\\_rdc\\_interrupts](#) { [kRDC\\_RestoreCompleteInterrupt](#) = RDC\_INTCTRL\_RCI\_EN\_MASK }  
*RDC interrupts.*
- enum [\\_rdc\\_flags](#) { [kRDC\\_PowerDownDomainOn](#) = RDC\_STAT\_PDS\_MASK }  
*RDC status.*
- 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)  
*Initializes the RDC module.*
- void [RDC\\_Deinit](#) (RDC\_Type \*base)  
*De-initializes the RDC module.*
- 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.*
- void [RDC\\_SetMasterDomainAssignment](#) (RDC\_Type \*base, rdc\_master\_t master, const [rdc\\_domain\\_assignment\\_t](#) \*domainAssignment)  
*Set master domain assignment.*
- void [RDC\\_GetDefaultMasterDomainAssignment](#) ([rdc\\_domain\\_assignment\\_t](#) \*domainAssignment)  
*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.*
- static uint8\_t [RDC\\_GetPeriphAccessPolicy](#) (RDC\_Type \*base, rdc\_periph\_t periph, uint8\_t domainId)  
*Get the peripheral access policy for specific domain.*
- 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\\_GetMemAccessPolicy](#) (RDC\_Type \*base, rdc\_mem\_t mem, uint8\_t domainId)  
*Get the memory region access policy for specific domain.*
- static uint8\_t [RDC\\_GetCurrentMasterDomainId](#) (RDC\_Type \*base)  
*Gets the domain ID of the current bus master.*

## 17.2 Data Structure Documentation

### 17.2.1 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.*

#### Field Documentation

- (1) uint32\_t rdc\_hardware\_config\_t::domainNumber
- (2) uint32\_t rdc\_hardware\_config\_t::masterNumber
- (3) uint32\_t rdc\_hardware\_config\_t::periphNumber
- (4) uint32\_t rdc\_hardware\_config\_t::memNumber

### 17.2.2 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.*

#### Field Documentation

- (1) uint32\_t rdc\_domain\_assignment\_t::domainId
- (2) uint32\_t rdc\_domain\_assignment\_t::\_\_pad0\_\_
- (3) uint32\_t rdc\_domain\_assignment\_t::lock

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

#### Data Fields

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

- *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.*

### Field Documentation

- (1) `rdc_periph_t rdc_periph_access_config_t::periph`
- (2) `bool rdc_periph_access_config_t::lock`
- (3) `bool rdc_periph_access_config_t::enableSema`
- (4) `uint16_t rdc_periph_access_config_t::policy`

### 17.2.4 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.*
- uint64\_t [baseAddress](#)  
*Start address of the memory region.*
- uint64\_t [endAddress](#)  
*End address of the memory region.*
- uint16\_t [policy](#)  
*Access policy.*

## Field Documentation

- (1) `rdc_mem_t rdc_mem_access_config_t::mem`
- (2) `bool rdc_mem_access_config_t::lock`
- (3) `uint64_t rdc_mem_access_config_t::baseAddress`
- (4) `uint64_t rdc_mem_access_config_t::endAddress`
- (5) `uint16_t rdc_mem_access_config_t::policy`

17.2.5 struct `rdc_mem_status_t`

## Data Fields

- `bool hasViolation`  
*Violating happens or not.*
- `uint8_t domainID`  
*Violating Domain ID.*
- `uint64_t address`  
*Violating Address.*

## Field Documentation

- (1) `bool rdc_mem_status_t::hasViolation`
- (2) `uint8_t rdc_mem_status_t::domainID`
- (3) `uint64_t rdc_mem_status_t::address`

## 17.3 Enumeration Type Documentation

17.3.1 enum `_rdc_interrupts`

Enumerator

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

17.3.2 enum `_rdc_flags`

Enumerator

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

### 17.3.3 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.

## 17.4 Function Documentation

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

This function enables the RDC clock.

Parameters

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

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

This function disables the RDC clock.

Parameters

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

### 17.4.3 void RDC\_GetHardwareConfig ( RDC\_Type \* *base*, rdc\_hardware\_config\_t \* *config* )

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. |

### 17.4.4 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> . |

#### 17.4.5 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> . |

#### 17.4.6 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](#).

#### 17.4.7 static void RDC\_ClearInterruptStatus ( RDC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## 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> . |

#### 17.4.8 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](#).

#### 17.4.9 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> . |

#### 17.4.10 void RDC\_SetMasterDomainAssignment ( RDC\_Type \* *base*, rdc\_master\_t *master*, const rdc\_domain\_assignment\_t \* *domainAssignment* )

## Parameters

|                          |                              |
|--------------------------|------------------------------|
| <i>base</i>              | RDC peripheral base address. |
| <i>master</i>            | Which master to set.         |
| <i>domain-Assignment</i> | Pointer to the assignment.   |

#### 17.4.11 void RDC\_GetDefaultMasterDomainAssignment ( rdc\_domain\_assignment\_t \* *domainAssignment* )

The default configuration is:

```
assignment->domainId = 0U;
assignment->lock = 0U;
```

## Parameters

|                          |                            |
|--------------------------|----------------------------|
| <i>domain-Assignment</i> | Pointer to the assignment. |
|--------------------------|----------------------------|

#### 17.4.12 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.        |

#### 17.4.13 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. |

#### 17.4.14 void RDC\_GetDefaultPeriphAccessConfig ( rdc\_periph\_access\_config\_t \* *config* )

The default configuration is:

```
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. |
|---------------|--------------------------------------|

**17.4.15 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.    |

**17.4.16 static uint8\_t RDC\_GetPeriphAccessPolicy ( RDC\_Type \* *base*, rdc\_periph\_t *periph*, uint8\_t *domainId* ) [inline], [static]**

## Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | RDC peripheral base address. |
| <i>periph</i>   | Which peripheral to get.     |
| <i>domainId</i> | Get policy for which domain. |

## Returns

Access policy, see [\\_rdc\\_access\\_policy](#).

**17.4.17 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. |

#### 17.4.18 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. |
|---------------|--------------------------------------|

#### 17.4.19 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. |

#### 17.4.20 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. |

#### 17.4.21 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. |
| <i>mem</i>    | Which memory region to get.  |
| <i>status</i> | The returned status.         |

#### 17.4.22 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. |

#### 17.4.23 static uint8\_t RDC\_GetMemAccessPolicy ( RDC\_Type \* *base*, rdc\_mem\_t *mem*, uint8\_t *domainId* ) [inline], [static]

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | RDC peripheral base address. |
| <i>mem</i>      | Which memory region to get.  |
| <i>domainId</i> | Get policy for which domain. |

Returns

Access policy, see [\\_rdc\\_access\\_policy](#).

#### 17.4.24 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 18

# RDC\_SEMA42: Hardware Semaphores Driver

### 18.1 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) (((volatile uint8\_t \*)&((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.*

- 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, 4))  
*RDC\_SEMA42 driver version.*

## 18.2 Macro Definition Documentation

### 18.2.1 #define RDC\_SEMA42\_GATE\_NUM\_RESET\_ALL (64U)

### 18.2.2 #define RDC\_SEMA42\_GATEn( base, n ) (((volatile uint8\_t \*)(&((base)->GATE0)))[(n)])

### 18.2.3 #define RDC\_SEMA42\_GATE\_COUNT (64U)

## 18.3 Function Documentation

### 18.3.1 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. |
|-------------|-------------------------------------|

### 18.3.2 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. |
|-------------|-------------------------------------|

### 18.3.3 **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. |

### 18.3.4 **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.        |

### 18.3.5 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.              |

### 18.3.6 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.

### 18.3.7 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.

### 18.3.8 `status_t RDC_SEMA42_ResetGate ( RDC_SEMAPHORE_Type * base, uint8_t gateNum )`

This function resets a RDC\_SEMA42 gate to an unlocked status.

## 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.   |

### 18.3.9 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 19

## SAI: Serial Audio Interface

### 19.1 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.

### 19.2 Typical configurations

#### Bit width configuration

SAI driver support 8/16/24/32bits stereo/mono raw audio data transfer. SAI EDMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI DMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI SDMA driver support 8/16/24/32bits stereo/mono raw audio data transfer.

#### Frame configuration

SAI driver support I2S, DSP, Left justified, Right justified, TDM mode. Application can call the api directly: `SAI_GetClassicI2SConfig` `SAI_GetLeftJustifiedConfig` `SAI_GetRightJustifiedConfig` `SAI_GetTDMConfig` `SAI_GetDSPConfig`

## 19.3 Typical use case

### 19.3.1 SAI Send/receive using an interrupt method

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

### 19.3.2 SAI Send/receive using a DMA method

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

## Modules

- [SAI Driver](#)
- [SAI SDMA Driver](#)

## 19.4 *Typical use case*

## 19.5 SAI Driver

### 19.5.1 Overview

#### Data Structures

- struct [sai\\_config\\_t](#)  
*SAI user configuration structure. [More...](#)*
- struct [sai\\_transfer\\_format\\_t](#)  
*sai transfer format [More...](#)*
- struct [sai\\_bit\\_clock\\_t](#)  
*sai bit clock configurations [More...](#)*
- struct [sai\\_frame\\_sync\\_t](#)  
*sai frame sync configurations [More...](#)*
- struct [sai\\_serial\\_data\\_t](#)  
*sai serial data configurations [More...](#)*
- struct [sai\\_transceiver\\_t](#)  
*sai transceiver configurations [More...](#)*
- struct [sai\\_transfer\\_t](#)  
*SAI transfer structure. [More...](#)*
- struct [sai\\_handle\\_t](#)  
*SAI handle structure. [More...](#)*

#### Macros

- #define [SAI\\_XFER\\_QUEUE\\_SIZE](#) (4U)  
*SAI transfer queue size, user can refine it according to use case.*
- #define [FSL\\_SAI\\_HAS\\_FIFO\\_EXTEND\\_FEATURE](#) 0  
*sai fifo feature*

#### 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 {  
[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\_status\_t, SAI return status.*
  - enum {
    - 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 }
  - \_sai\_channel\_mask, sai channel mask value, actual channel numbers is depend soc specific*
  - 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,
      - kSAI\_Bclk\_Master\_FrameSync\_Slave = 0x2U,
      - kSAI\_Bclk\_Slave\_FrameSync\_Master = 0x3U }
      - 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 = 0x1U,
            - kSAI\_SampleOnFallingEdge = 0x0U,
            - kSAI\_SampleOnRisingEdge = 0x1U }
            - SAI clock polarity, active high or low.*
            - enum sai\_sync\_mode\_t {
              - kSAI\_ModeAsync = 0x0U,
              - kSAI\_ModeSync }
              - 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 {  
kSAI\_WordStartInterruptEnable,  
kSAI\_SyncErrorInterruptEnable = I2S\_TCSR\_SEIE\_MASK,  
kSAI\_FIFOWarningInterruptEnable = I2S\_TCSR\_FWIE\_MASK,  
kSAI\_FIFOErrorInterruptEnable = I2S\_TCSR\_FEIE\_MASK }  
\_sai\_interrupt\_enable\_t, *The SAI interrupt enable flag*
- enum { kSAI\_FIFOWarningDMAEnable = I2S\_TCSR\_FWDE\_MASK }  
\_sai\_dma\_enable\_t, *The DMA request sources*
- enum {  
kSAI\_WordStartFlag = I2S\_TCSR\_WSF\_MASK,  
kSAI\_SyncErrorFlag = I2S\_TCSR\_SEF\_MASK,  
kSAI\_FIFOErrorFlag = I2S\_TCSR\_FEF\_MASK,  
kSAI\_FIFOWarningFlag = I2S\_TCSR\_FWF\_MASK }  
\_sai\_flags, *The SAI status flag*
- enum sai\_reset\_type\_t {  
kSAI\_ResetTypeSoftware = I2S\_TCSR\_SR\_MASK,  
kSAI\_ResetTypeFIFO = I2S\_TCSR\_FR\_MASK,  
kSAI\_ResetAll = I2S\_TCSR\_SR\_MASK | I2S\_TCSR\_FR\_MASK }  
*The reset type.*
- enum sai\_sample\_rate\_t {  
kSAI\_SampleRate8KHz = 8000U,  
kSAI\_SampleRate11025Hz = 11025U,  
kSAI\_SampleRate12KHz = 12000U,  
kSAI\_SampleRate16KHz = 16000U,  
kSAI\_SampleRate22050Hz = 22050U,  
kSAI\_SampleRate24KHz = 24000U,  
kSAI\_SampleRate32KHz = 32000U,  
kSAI\_SampleRate44100Hz = 44100U,  
kSAI\_SampleRate48KHz = 48000U,  
kSAI\_SampleRate96KHz = 96000U,  
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.
• enum sai_transceiver_type_t {
 kSAI_Transmitter = 0U,
 kSAI_Receiver = 1U }
 sai transceiver type
• enum sai_frame_sync_len_t {
 kSAI_FrameSyncLenOneBitClk = 0U,
 kSAI_FrameSyncLenPerWordWidth = 1U }
 sai frame sync len

```

## Driver version

- #define FSL\_SAI\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 8))  
*Version 2.3.8.*

## 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\_Init (I2S\_Type \*base)  
*Initializes the SAI peripheral.*
- 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.*
- static void SAI\_TxSetBitClockDirection (I2S\_Type \*base, sai\_master\_slave\_t masterSlave)  
*Set Rx bit clock direction.*
- static void SAI\_RxSetBitClockDirection (I2S\_Type \*base, sai\_master\_slave\_t masterSlave)  
*Set Rx bit clock direction.*
- static void SAI\_RxSetFrameSyncDirection (I2S\_Type \*base, sai\_master\_slave\_t masterSlave)  
*Set Rx frame sync direction.*

- static void [SAI\\_TxSetFrameSyncDirection](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave)  
*Set Tx frame sync direction.*
- void [SAI\\_TxSetBitClockRate](#) (I2S\_Type \*base, uint32\_t sourceClockHz, uint32\_t sampleRate, uint32\_t bitWidth, uint32\_t channelNumbers)  
*Transmitter bit clock rate configurations.*
- void [SAI\\_RxSetBitClockRate](#) (I2S\_Type \*base, uint32\_t sourceClockHz, uint32\_t sampleRate, uint32\_t bitWidth, uint32\_t channelNumbers)  
*Receiver bit clock rate configurations.*
- void [SAI\\_TxSetBitclockConfig](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave, [sai\\_bit\\_clock\\_t](#) \*config)  
*Transmitter Bit clock configurations.*
- void [SAI\\_RxSetBitclockConfig](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave, [sai\\_bit\\_clock\\_t](#) \*config)  
*Receiver Bit clock configurations.*
- void [SAI\\_TxSetFrameSyncConfig](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave, [sai\\_frame\\_sync\\_t](#) \*config)  
*SAI transmitter Frame sync configurations.*
- void [SAI\\_RxSetFrameSyncConfig](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave, [sai\\_frame\\_sync\\_t](#) \*config)  
*SAI receiver Frame sync configurations.*
- void [SAI\\_TxSetSerialDataConfig](#) (I2S\_Type \*base, [sai\\_serial\\_data\\_t](#) \*config)  
*SAI transmitter Serial data configurations.*
- void [SAI\\_RxSetSerialDataConfig](#) (I2S\_Type \*base, [sai\\_serial\\_data\\_t](#) \*config)  
*SAI receiver Serial data configurations.*
- void [SAI\\_TxSetConfig](#) (I2S\_Type \*base, [sai\\_transceiver\\_t](#) \*config)  
*SAI transmitter configurations.*
- void [SAI\\_RxSetConfig](#) (I2S\_Type \*base, [sai\\_transceiver\\_t](#) \*config)  
*SAI receiver configurations.*
- void [SAI\\_GetClassicI2SConfig](#) ([sai\\_transceiver\\_t](#) \*config, [sai\\_word\\_width\\_t](#) bitWidth, [sai\\_mono\\_stereo\\_t](#) mode, uint32\_t saiChannelMask)  
*Get classic I2S mode configurations.*
- void [SAI\\_GetLeftJustifiedConfig](#) ([sai\\_transceiver\\_t](#) \*config, [sai\\_word\\_width\\_t](#) bitWidth, [sai\\_mono\\_stereo\\_t](#) mode, uint32\_t saiChannelMask)  
*Get left justified mode configurations.*
- void [SAI\\_GetRightJustifiedConfig](#) ([sai\\_transceiver\\_t](#) \*config, [sai\\_word\\_width\\_t](#) bitWidth, [sai\\_mono\\_stereo\\_t](#) mode, uint32\_t saiChannelMask)  
*Get right justified mode configurations.*
- void [SAI\\_GetTDMConfig](#) ([sai\\_transceiver\\_t](#) \*config, [sai\\_frame\\_sync\\_len\\_t](#) frameSyncWidth, [sai\\_word\\_width\\_t](#) bitWidth, uint32\_t dataWordNum, uint32\_t saiChannelMask)  
*Get TDM mode configurations.*
- void [SAI\\_GetDSPConfig](#) ([sai\\_transceiver\\_t](#) \*config, [sai\\_frame\\_sync\\_len\\_t](#) frameSyncWidth, [sai\\_word\\_width\\_t](#) bitWidth, [sai\\_mono\\_stereo\\_t](#) mode, uint32\_t saiChannelMask)  
*Get DSP mode configurations.*

## 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 Tx 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 resetType)  
*Do software reset or FIFO reset.*
- void [SAI\\_RxSoftwareReset](#) (I2S\_Type \*base, sai\_reset\_type\_t resetType)  
*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.*

## 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 uintptr\_t [SAI\\_TxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Tx data register address.*
- static uintptr\_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.*
- void **SAI\_TransferTxSetConfig** (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transceiver\_t \*config)  
*SAI transmitter transfer configurations.*
- void **SAI\_TransferRxSetConfig** (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transceiver\_t \*config)  
*SAI receiver transfer configurations.*
- 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.*

## 19.5.2 Data Structure Documentation

### 19.5.2.1 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.*

### 19.5.2.2 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.*
- [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.*

## Field Documentation

### (1) bool sai\_transfer\_format\_t::isFrameSyncCompact

#### 19.5.2.3 struct sai\_bit\_clock\_t

## Data Fields

- bool [bclkSrcSwap](#)  
*bit clock source swap*
- bool [bclkInputDelay](#)  
*bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .*
- sai\_clock\_polarity\_t [bclkPolarity](#)  
*bit clock polarity*
- sai\_bclk\_source\_t [bclkSource](#)  
*bit Clock source*

## Field Documentation

### (1) bool sai\_bit\_clock\_t::bclkInputDelay

#### 19.5.2.4 struct sai\_frame\_sync\_t

## Data Fields

- uint8\_t [frameSyncWidth](#)  
*frame sync width in number of bit clocks*
- bool [frameSyncEarly](#)  
*TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.*
- sai\_clock\_polarity\_t [frameSyncPolarity](#)  
*frame sync polarity*

#### 19.5.2.5 struct sai\_serial\_data\_t

## Data Fields

- sai\_data\_order\_t [dataOrder](#)  
*configure whether the LSB or MSB is transmitted first*
- uint8\_t [dataWord0Length](#)



- *configure the number of bits in the first word in each frame*  
uint8\_t [dataWordNLength](#)
- *configure the number of bits in the each word in each frame, except the first word*  
uint8\_t [dataWordLength](#)
- *used to record the data length for dma transfer*  
uint8\_t [dataFirstBitShifted](#)
- *Configure the bit index for the first bit transmitted for each word in the frame.*  
uint8\_t [dataWordNum](#)
- *configure the number of words in each frame*  
uint32\_t [dataMaskedWord](#)
- *configure whether the transmit word is masked*

### 19.5.2.6 struct sai\_transceiver\_t

#### Data Fields

- [sai\\_serial\\_data\\_t](#) serialData  
*serial data configurations*
- [sai\\_frame\\_sync\\_t](#) frameSync  
*ws configurations*
- [sai\\_bit\\_clock\\_t](#) bitClock  
*bit clock configurations*
- [sai\\_master\\_slave\\_t](#) masterSlave  
*transceiver is master or slave*
- [sai\\_sync\\_mode\\_t](#) syncMode  
*transceiver sync mode*
- uint8\_t [startChannel](#)  
*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.*

### 19.5.2.7 struct sai\_transfer\_t

#### Data Fields

- uint8\_t \* [data](#)  
*Data start address to transfer.*
- size\_t [dataSize](#)  
*Transfer size.*



## Field Documentation

(1) `uint8_t* sai_transfer_t::data`

(2) `size_t sai_transfer_t::dataSize`

### 19.5.2.8 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.*

## 19.5.3 Macro Definition Documentation

### 19.5.3.1 #define SAI\_XFER\_QUEUE\_SIZE (4U)

## 19.5.4 Enumeration Type Documentation

### 19.5.4.1 anonymous enum

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.

#### 19.5.4.2 anonymous enum

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

#### 19.5.4.3 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.

#### 19.5.4.4 enum sai\_master\_slave\_t

Enumerator

***kSAI\_Master*** Master mode include bclk and frame sync.  
***kSAI\_Slave*** Slave mode include bclk and frame sync.  
***kSAI\_Bclk\_Master\_FrameSync\_Slave*** bclk in master mode, frame sync in slave mode  
***kSAI\_Bclk\_Slave\_FrameSync\_Master*** bclk in slave mode, frame sync in master mode

#### 19.5.4.5 enum sai\_mono\_stereo\_t

Enumerator

***kSAI\_Stereo*** Stereo sound.

***kSAI\_MonoRight*** Only Right channel have sound.

***kSAI\_MonoLeft*** Only left channel have sound.

#### 19.5.4.6 enum sai\_data\_order\_t

Enumerator

***kSAI\_DataLSB*** LSB bit transferred first.

***kSAI\_DataMSB*** MSB bit transferred first.

#### 19.5.4.7 enum sai\_clock\_polarity\_t

Enumerator

***kSAI\_PolarityActiveHigh*** Drive outputs on rising edge.

***kSAI\_PolarityActiveLow*** Drive outputs on falling edge.

***kSAI\_SampleOnFallingEdge*** Sample inputs on falling edge.

***kSAI\_SampleOnRisingEdge*** Sample inputs on rising edge.

#### 19.5.4.8 enum sai\_sync\_mode\_t

Enumerator

***kSAI\_ModeAsync*** Asynchronous mode.

***kSAI\_ModeSync*** Synchronous mode (with receiver or transmit)

#### 19.5.4.9 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.

#### 19.5.4.10 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.

#### 19.5.4.11 anonymous enum

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.

#### 19.5.4.12 anonymous enum

Enumerator

***kSAI\_FIFOWarningDMAEnable*** FIFO warning caused by the DMA request.

#### 19.5.4.13 anonymous enum

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\_FIFOWarningFlag*** FIFO warning flag.

#### 19.5.4.14 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.

#### 19.5.4.15 enum sai\_sample\_rate\_t

Enumerator

***kSAI\_SampleRate8KHz*** Sample rate 8000 Hz.  
***kSAI\_SampleRate11025Hz*** Sample rate 11025 Hz.

*kSAI\_SampleRate12KHz* Sample rate 12000 Hz.  
*kSAI\_SampleRate16KHz* Sample rate 16000 Hz.  
*kSAI\_SampleRate22050Hz* Sample rate 22050 Hz.  
*kSAI\_SampleRate24KHz* Sample rate 24000 Hz.  
*kSAI\_SampleRate32KHz* Sample rate 32000 Hz.  
*kSAI\_SampleRate44100Hz* Sample rate 44100 Hz.  
*kSAI\_SampleRate48KHz* Sample rate 48000 Hz.  
*kSAI\_SampleRate96KHz* Sample rate 96000 Hz.  
*kSAI\_SampleRate192KHz* Sample rate 192000 Hz.  
*kSAI\_SampleRate384KHz* Sample rate 384000 Hz.

#### 19.5.4.16 enum sai\_word\_width\_t

Enumerator

*kSAI\_WordWidth8bits* Audio data width 8 bits.  
*kSAI\_WordWidth16bits* Audio data width 16 bits.  
*kSAI\_WordWidth24bits* Audio data width 24 bits.  
*kSAI\_WordWidth32bits* Audio data width 32 bits.

#### 19.5.4.17 enum sai\_transceiver\_type\_t

Enumerator

*kSAI\_Transmitter* sai transmitter  
*kSAI\_Receiver* sai receiver

#### 19.5.4.18 enum sai\_frame\_sync\_len\_t

Enumerator

*kSAI\_FrameSyncLenOneBitClk* 1 bit clock frame sync len for DSP mode  
*kSAI\_FrameSyncLenPerWordWidth* Frame sync length decided by word width.

### 19.5.5 Function Documentation

#### 19.5.5.1 void SAI\_TxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )

**Deprecated** Do not use this function. It has been superceded by [SAI\\_Init](#)

Un-gates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_TxGetDefaultConfig\(\)](#).

## Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

### 19.5.5.2 void SAI\_RxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )

**Deprecated** Do not use this function. It has been superseded by [SAI\\_Init](#)

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. |

### 19.5.5.3 void SAI\_TxGetDefaultConfig ( sai\_config\_t \* *config* )

**Deprecated** Do not use this function. It has been superseded by [SAI\\_GetClassicI2SConfig](#), [SAI\\_GetLeftJustifiedConfig](#), [SAI\\_GetRightJustifiedConfig](#), [SAI\\_GetDSPConfig](#), [SAI\\_GetTDMConfig](#)

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 |
|---------------|-------------------------------------------|

**19.5.5.4 void SAI\_RxGetDefaultConfig ( sai\_config\_t \* *config* )**

**Deprecated** Do not use this function. It has been superceded by [SAI\\_GetClassicI2SConfig](#), [SAI\\_GetLeftJustifiedConfig](#) , [SAI\\_GetRightJustifiedConfig](#), [SAI\\_GetDSPConfig](#), [SAI\\_GetTDMConfig](#)

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 |
|---------------|-------------------------------------------|

**19.5.5.5 void SAI\_Init ( I2S\_Type \* *base* )**

This API gates the SAI clock. The SAI module can't operate unless SAI\_Init is called to enable the clock.

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|

**19.5.5.6 void SAI\_Deinit ( I2S\_Type \* *base* )**

This API gates the SAI clock. The SAI module can't operate unless SAI\_TxInit or SAI\_RxInit is called to enable the clock.

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|

**19.5.5.7 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 |
|-------------|------------------|

#### 19.5.5.8 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 |
|-------------|------------------|

#### 19.5.5.9 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. |

#### 19.5.5.10 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. |

#### 19.5.5.11 static void SAI\_TxSetBitClockDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]

Select bit clock direction, master or slave.

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SAI base pointer. |
|-------------|-------------------|



|                    |                               |
|--------------------|-------------------------------|
| <i>masterSlave</i> | reference sai_master_slave_t. |
|--------------------|-------------------------------|

**19.5.5.12 static void SAI\_RxSetBitClockDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]**

Select bit clock direction, master or slave.

Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | SAI base pointer.             |
| <i>masterSlave</i> | reference sai_master_slave_t. |

**19.5.5.13 static void SAI\_RxSetFrameSyncDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]**

Select frame sync direction, master or slave.

Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | SAI base pointer.             |
| <i>masterSlave</i> | reference sai_master_slave_t. |

**19.5.5.14 static void SAI\_TxSetFrameSyncDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]**

Select frame sync direction, master or slave.

Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | SAI base pointer.             |
| <i>masterSlave</i> | reference sai_master_slave_t. |

**19.5.5.15 void SAI\_TxSetBitClockRate ( I2S\_Type \* *base*, uint32\_t *sourceClockHz*, uint32\_t *sampleRate*, uint32\_t *bitWidth*, uint32\_t *channelNumbers* )**

## Parameters

|                        |                             |
|------------------------|-----------------------------|
| <i>base</i>            | SAI base pointer.           |
| <i>sourceClockHz</i>   | Bit clock source frequency. |
| <i>sampleRate</i>      | Audio data sample rate.     |
| <i>bitWidth</i>        | Audio data bitWidth.        |
| <i>channel-Numbers</i> | Audio channel numbers.      |

**19.5.5.16 void SAI\_RxSetBitClockRate ( I2S\_Type \* *base*, uint32\_t *sourceClockHz*, uint32\_t *sampleRate*, uint32\_t *bitWidth*, uint32\_t *channelNumbers* )**

## Parameters

|                        |                             |
|------------------------|-----------------------------|
| <i>base</i>            | SAI base pointer.           |
| <i>sourceClockHz</i>   | Bit clock source frequency. |
| <i>sampleRate</i>      | Audio data sample rate.     |
| <i>bitWidth</i>        | Audio data bitWidth.        |
| <i>channel-Numbers</i> | Audio channel numbers.      |

**19.5.5.17 void SAI\_TxSetBitclockConfig ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave*, sai\_bit\_clock\_t \* *config* )**

## Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                          |
| <i>masterSlave</i> | master or slave.                                           |
| <i>config</i>      | bit clock other configurations, can be NULL in slave mode. |

**19.5.5.18 void SAI\_RxSetBitclockConfig ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave*, sai\_bit\_clock\_t \* *config* )**

## Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                          |
| <i>masterSlave</i> | master or slave.                                           |
| <i>config</i>      | bit clock other configurations, can be NULL in slave mode. |

**19.5.5.19 void SAI\_TxSetFrameSyncConfig ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave*, sai\_frame\_sync\_t \* *config* )**

## Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                     |
| <i>masterSlave</i> | master or slave.                                      |
| <i>config</i>      | frame sync configurations, can be NULL in slave mode. |

**19.5.5.20 void SAI\_RxSetFrameSyncConfig ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave*, sai\_frame\_sync\_t \* *config* )**

## Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                     |
| <i>masterSlave</i> | master or slave.                                      |
| <i>config</i>      | frame sync configurations, can be NULL in slave mode. |

**19.5.5.21 void SAI\_TxSetSerialDataConfig ( I2S\_Type \* *base*, sai\_serial\_data\_t \* *config* )**

## Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | serial data configurations. |

**19.5.5.22 void SAI\_RxSetSerialDataConfig ( I2S\_Type \* *base*, sai\_serial\_data\_t \* *config* )**

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | serial data configurations. |

**19.5.5.23 void SAI\_TxSetConfig ( I2S\_Type \* *base*, sai\_transceiver\_t \* *config* )**

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | transmitter configurations. |

**19.5.5.24 void SAI\_RxSetConfig ( I2S\_Type \* *base*, sai\_transceiver\_t \* *config* )**

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>config</i> | receiver configurations. |

**19.5.5.25 void SAI\_GetClassicI2SConfig ( sai\_transceiver\_t \* *config*, sai\_word\_width\_t *bitWidth*, sai\_mono\_stereo\_t *mode*, uint32\_t *saiChannelMask* )**

Parameters

|                       |                                         |
|-----------------------|-----------------------------------------|
| <i>config</i>         | transceiver configurations.             |
| <i>bitWidth</i>       | audio data bitWidth.                    |
| <i>mode</i>           | audio data channel.                     |
| <i>saiChannelMask</i> | mask value of the channel to be enable. |

**19.5.5.26 void SAI\_GetLeftJustifiedConfig ( sai\_transceiver\_t \* *config*, sai\_word\_width\_t *bitWidth*, sai\_mono\_stereo\_t *mode*, uint32\_t *saiChannelMask* )**

## Parameters

|                        |                                         |
|------------------------|-----------------------------------------|
| <i>config</i>          | transceiver configurations.             |
| <i>bitWidth</i>        | audio data bitWidth.                    |
| <i>mode</i>            | audio data channel.                     |
| <i>saiChannel-Mask</i> | mask value of the channel to be enable. |

**19.5.5.27 void SAI\_GetRightJustifiedConfig ( sai\_transceiver\_t \* *config*,  
sai\_word\_width\_t *bitWidth*, sai\_mono\_stereo\_t *mode*, uint32\_t *saiChannelMask*  
)**

## Parameters

|                        |                                         |
|------------------------|-----------------------------------------|
| <i>config</i>          | transceiver configurations.             |
| <i>bitWidth</i>        | audio data bitWidth.                    |
| <i>mode</i>            | audio data channel.                     |
| <i>saiChannel-Mask</i> | mask value of the channel to be enable. |

**19.5.5.28 void SAI\_GetTDMConfig ( sai\_transceiver\_t \* *config*, sai\_frame\_sync\_len\_t  
*frameSyncWidth*, sai\_word\_width\_t *bitWidth*, uint32\_t *dataWordNum*, uint32\_t  
*saiChannelMask* )**

## Parameters

|                        |                                         |
|------------------------|-----------------------------------------|
| <i>config</i>          | transceiver configurations.             |
| <i>frameSync-Width</i> | length of frame sync.                   |
| <i>bitWidth</i>        | audio data word width.                  |
| <i>dataWordNum</i>     | word number in one frame.               |
| <i>saiChannel-Mask</i> | mask value of the channel to be enable. |

**19.5.5.29 void SAI\_GetDSPConfig ( sai\_transceiver\_t \* *config*, sai\_frame\_sync\_len\_t *frameSyncWidth*, sai\_word\_width\_t *bitWidth*, sai\_mono\_stereo\_t *mode*, uint32\_t *saiChannelMask* )**

Note

DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI\_RxSetConfig instead of SAI\_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
 kSAI_Stereo, channelMask)
* config->frameSync.frameSyncEarly = true;
* SAI_TxSetConfig(base, config)
*
```

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI\_RxSetConfig instead of SAI\_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
 kSAI_Stereo, channelMask)
* SAI_TxSetConfig(base, config)
*
```

Parameters

|                        |                                      |
|------------------------|--------------------------------------|
| <i>config</i>          | transceiver configurations.          |
| <i>frameSync-Width</i> | length of frame sync.                |
| <i>bitWidth</i>        | audio data bitWidth.                 |
| <i>mode</i>            | audio data channel.                  |
| <i>saiChannel-Mask</i> | mask value of the channel to enable. |

**19.5.5.30 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.

**19.5.5.31 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> |

**19.5.5.32** `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.

**19.5.5.33** `static void SAI_RxClearStatusFlags ( I2S_Type * base, uint32_t mask ) [inline], [static]`

## Parameters

|             |                                                                                                                                                                                                             |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                            |
| <i>mask</i> | State mask. It can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_WordStartFlag</li> <li>• kSAI_SyncErrorFlag</li> <li>• kSAI_FIFOErrorFlag</li> </ul> |

#### 19.5.5.34 void SAI\_TxSoftwareReset ( I2S\_Type \* *base*, sai\_reset\_type\_t *resetType* )

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>resetType</i> | Reset type, FIFO reset or software reset |

#### 19.5.5.35 void SAI\_RxSoftwareReset ( I2S\_Type \* *base*, sai\_reset\_type\_t *resetType* )

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>resetType</i> | Reset type, FIFO reset or software reset |

#### 19.5.5.36 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. |

#### 19.5.5.37 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. |

**19.5.5.38 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 |

**19.5.5.39 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 |

**19.5.5.40 void SAI\_TxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

## Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**19.5.5.41 void SAI\_RxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

## Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**19.5.5.42** void SAI\_TxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

## Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

#### 19.5.5.43 void SAI\_RxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )

## Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

#### 19.5.5.44 static void SAI\_TxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                                                                                                                                                                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                    |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFOResourceInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**19.5.5.45** `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> |

**19.5.5.46** `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> |

**19.5.5.47** `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_FIFOResultInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**19.5.5.48 static void SAI\_TxEnableDMA ( I2S\_Type \* *base*, uint32\_t *mask*, bool *enable* )**  
**[inline], [static]**

## Parameters

|               |                                                                                                                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer                                                                                                                                                                                   |
| <i>mask</i>   | DMA source The parameter can be combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_FIFOWarningDMAEnable</li> <li>• kSAI_FIFOResultDMAEnable</li> </ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.                                                                                                                                                    |

**19.5.5.49 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. |
|---------------|-------------------------------------------------|

#### 19.5.5.50 static uintptr\_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.

#### 19.5.5.51 static uintptr\_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.

#### 19.5.5.52 void SAI\_TxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )

**Deprecated** Do not use this function. It has been superseded by [SAI\\_TxSetConfig](#)

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. |

**19.5.5.53 void SAI\_RxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

**Deprecated** Do not use this function. It has been superseded by [SAI\\_RxSetConfig](#)

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. |

**19.5.5.54 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.                                     |

**19.5.5.55 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.                                     |

**19.5.5.56 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. |

**19.5.5.57 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.                                        |

**19.5.5.58 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.                                        |

**19.5.5.59 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.

**19.5.5.60 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 |

#### 19.5.5.61 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. |

#### 19.5.5.62 void SAI\_TransferTxSetConfig ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transceiver\_t \* *config* )

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

## Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>handle</i> | SAI handle pointer.         |
| <i>config</i> | transmitter configurations. |

#### 19.5.5.63 void SAI\_TransferRxSetConfig ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transceiver\_t \* *config* )

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI handle pointer.      |
| <i>config</i> | receiver configurations. |

**19.5.5.64** `status_t SAI_TransferTxSetFormat ( I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

**Deprecated** Do not use this function. It has been superceded by [SAI\\_TransferTxSetConfig](#)

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.

**19.5.5.65** `status_t SAI_TransferRxSetFormat ( I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

**Deprecated** Do not use this function. It has been superceded by [SAI\\_TransferRxSetConfig](#)

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.

#### 19.5.5.66 status\_t SAI\_TransferSendNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This API returns immediately after the transfer initiates. Call the SAI\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <a href="#">sai_transfer_t</a> structure.               |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Successfully started the data receive. |
| <i>kStatus_SAI_TxBusy</i>      | Previous receive still not finished.   |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.        |

#### 19.5.5.67 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.        |

#### 19.5.5.68 status\_t SAI\_TransferGetSendCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count sent.                                                      |

## Return values

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

#### 19.5.5.69 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. |

**19.5.5.70 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. |

**19.5.5.71 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. |

**19.5.5.72 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. |

**19.5.5.73 void SAI\_TransferTerminateReceive ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )**

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI\_TransferAbortReceive.

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |

**19.5.5.74 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. |

**19.5.5.75 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. |



## 19.6 SAI SDMA Driver

### 19.6.1 Typical use case

### 19.6.2 Overview

#### Multi fifo transfer use sai sdma driver

The SDMA multi fifo script support transfer data between multi peripheral fifos and memory, a typical user case is that receiving multi sai channel data and put it into memory as

| channel 0 | channel 1 | channel 2 | channel 3 | channel 4 | ..... |

Multi fifo script is target to implement above feature, it can supports 1.configurable fifo watermark range from  $1 \sim (2^{12}-1)$ , it is a value of `fifo_watermark * channel_numbers` 2.configurable fifo numbers, support up to 15 continuous fifos 3.configurable fifo address offset, support address offset up to 64

```
/* load sdma script */
SDMA_LoadScript()
/* sai multi channel configurations */
SAI_GetClassicI2SConfig(&config, DEMO_AUDIO_BIT_WIDTH, kSAI_Stereo,
 kSAI_Channel0Mask | kSAI_Channel1Mask |
 kSAI_Channel2Mask | kSAI_Channel3Mask | kSAI_Channel4Mask);
SAI_TransferRxSetConfigSDMA(SAI, handle, &config);
SAI_TransferReceiveSDMA(SAI, handle, &config);
```

Transmitting data using multi fifo is same as above.

### Data Structures

- struct [sai\\_sdma\\_handle\\_t](#)  
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

### Typedefs

- typedef void(\* [sai\\_sdma\\_callback\\_t](#))(I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
SAI SDMA transfer callback function for finish and error.

### Driver version

- #define [FSL\\_SAI\\_SDMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 5, 3))  
Version 2.5.3.

## SDMA Transactional

- void [SAI\\_TransferTxCreateHandleSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_sdma\\_callback\\_t](#) callback, void \*userData, [sdma\\_handle\\_t](#) \*dmaHandle, uint32\_t eventSource)  
*Initializes the SAI SDMA handle.*
- void [SAI\\_TransferRxCreateHandleSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_sdma\\_callback\\_t](#) callback, void \*userData, [sdma\\_handle\\_t](#) \*dmaHandle, uint32\_t eventSource)  
*Initializes the SAI Rx SDMA handle.*
- void [SAI\\_TransferTxSetFormatSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_TransferRxSetFormatSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- [status\\_t](#) [SAI\\_TransferSendSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- [status\\_t](#) [SAI\\_TransferReceiveSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI receive using SDMA.*
- void [SAI\\_TransferAbortSendSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle)  
*Aborts a SAI transfer using SDMA.*
- void [SAI\\_TransferAbortReceiveSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle)  
*Aborts a SAI receive using SDMA.*
- void [SAI\\_TransferTerminateReceiveSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle)  
*Terminate all the SAI sdma receive transfer.*
- void [SAI\\_TransferTerminateSendSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle)  
*Terminate all the SAI sdma send transfer.*
- void [SAI\\_TransferRxSetConfigSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transceiver\\_t](#) \*saiConfig)  
*brief Configures the SAI RX.*
- void [SAI\\_TransferTxSetConfigSDMA](#) (I2S\_Type \*base, sai\_sdma\_handle\_t \*handle, [sai\\_transceiver\\_t](#) \*saiConfig)  
*brief Configures the SAI Tx.*

### 19.6.3 Data Structure Documentation

#### 19.6.3.1 struct \_sai\_sdma\_handle

##### Data Fields

- [sdma\\_handle\\_t](#) \* [dmaHandle](#)  
*DMA handler for SAI send.*
- uint8\_t [bytesPerFrame](#)  
*Bytes in a frame.*
- uint8\_t [channel](#)  
*start data channel*
- uint8\_t [channelNums](#)

- *total transfer channel numbers, used for multifo*
- uint8\_t [channelMask](#)  
*enabled channel mask value, refernece \_sai\_channel\_mask*
- uint8\_t [fifoOffset](#)  
*fifo address offset between multifo*
- uint32\_t [count](#)  
*The transfer data count in a DMA request.*
- uint32\_t [state](#)  
*Internal state for SAI SDMA transfer.*
- uint32\_t [eventSource](#)  
*SAI event source number.*
- [sai\\_sdma\\_callback\\_t](#) [callback](#)  
*Callback for users while transfer finish or error occurs.*
- void \* [userData](#)  
*User callback parameter.*
- [sdma\\_buffer\\_descriptor\\_t](#) [bdPool](#) [SAI\_XFER\_QUEUE\_SIZE]  
*BD pool for SDMA transfer.*
- [sai\\_transfer\\_t](#) [saiQueue](#) [SAI\_XFER\_QUEUE\_SIZE]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [SAI\_XFER\_QUEUE\_SIZE]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*

## Field Documentation

- (1) [sdma\\_buffer\\_descriptor\\_t](#) [sai\\_sdma\\_handle\\_t::bdPool](#)[SAI\_XFER\_QUEUE\_SIZE]
- (2) [sai\\_transfer\\_t](#) [sai\\_sdma\\_handle\\_t::saiQueue](#)[SAI\_XFER\_QUEUE\_SIZE]
- (3) volatile uint8\_t [sai\\_sdma\\_handle\\_t::queueUser](#)

## 19.6.4 Function Documentation

**19.6.4.1** void [SAI\\_TransferTxCreateHandleSDMA](#) ( I2S\_Type \* *base*, [sai\\_sdma\\_handle\\_t](#) \* *handle*, [sai\\_sdma\\_callback\\_t](#) *callback*, void \* *userData*, [sdma\\_handle\\_t](#) \* *dmaHandle*, uint32\_t *eventSource* )

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

Parameters

---

|                    |                                                                      |
|--------------------|----------------------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                                    |
| <i>handle</i>      | SAI SDMA handle pointer.                                             |
| <i>base</i>        | SAI peripheral base address.                                         |
| <i>callback</i>    | Pointer to user callback function.                                   |
| <i>userData</i>    | User parameter passed to the callback function.                      |
| <i>dmaHandle</i>   | SDMA handle pointer, this handle shall be static allocated by users. |
| <i>eventSource</i> | SAI event source number.                                             |

**19.6.4.2 void SAI\_TransferRxCreateHandleSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_sdma\_callback\_t *callback*, void \* *userData*, sdma\_handle\_t \* *dmaHandle*, uint32\_t *eventSource* )**

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

Parameters

|                    |                                                                      |
|--------------------|----------------------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                                    |
| <i>handle</i>      | SAI SDMA handle pointer.                                             |
| <i>base</i>        | SAI peripheral base address.                                         |
| <i>callback</i>    | Pointer to user callback function.                                   |
| <i>userData</i>    | User parameter passed to the callback function.                      |
| <i>dmaHandle</i>   | SDMA handle pointer, this handle shall be static allocated by users. |
| <i>eventSource</i> | SAI event source number.                                             |

**19.6.4.3 void SAI\_TransferTxSetFormatSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

Parameters

|                           |                                                                                                                                 |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                               |
| <i>handle</i>             | SAI SDMA handle pointer.                                                                                                        |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                        |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**19.6.4.4 void SAI\_TransferRxSetFormatSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the SDMA parameter according to formatting requirements.

Parameters

|                           |                                                                                                                                      |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                    |
| <i>handle</i>             | SAI SDMA handle pointer.                                                                                                             |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                          |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                             |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format. |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**19.6.4.5 status\_t SAI\_TransferSendSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

This interface returns immediately after the transfer initiates. Call SAI\_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SAI base pointer.                      |
| <i>handle</i> | SAI SDMA handle pointer.               |
| <i>xfer</i>   | Pointer to the DMA transfer structure. |

## Return values

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI SDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.      |
| <i>kStatus_TxBusy</i>          | SAI is busy sending data.           |

#### 19.6.4.6 status\_t SAI\_TransferReceiveSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This interface returns immediately after the transfer initiates. Call the SAI\_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer                   |
| <i>handle</i> | SAI SDMA handle pointer.           |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI SDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.         |
| <i>kStatus_RxBusy</i>          | SAI is busy receiving data.            |

#### 19.6.4.7 void SAI\_TransferAbortSendSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle* )

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI SDMA handle pointer. |

**19.6.4.8 void SAI\_TransferAbortReceiveSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle* )**

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer         |
| <i>handle</i> | SAI SDMA handle pointer. |

**19.6.4.9 void SAI\_TransferTerminateReceiveSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle* )**

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI SDMA handle pointer. |

**19.6.4.10 void SAI\_TransferTerminateSendSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle* )**

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI SDMA handle pointer. |

**19.6.4.11 void SAI\_TransferRxSetConfigSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transceiver\_t \* *saiConfig* )**

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.

**19.6.4.12 void SAI\_TransferTxSetConfigSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transceiver\_t \* *saiConfig* )**

param base SAI base pointer. param handle SAI SDMA handle pointer. param saiConig sai configurations.



## Chapter 20

# SDMA: Smart Direct Memory Access (SDMA) Controller Driver

## 20.1 Overview

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

## 20.2 Typical use case

### 20.2.1 SDMA Operation

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

## Data Structures

- struct [sdma\\_config\\_t](#)  
*SDMA global configuration structure. [More...](#)*
- struct [sdma\\_multi\\_fifo\\_config\\_t](#)  
*SDMA multi fifo configurations. [More...](#)*
- struct [sdma\\_sw\\_done\\_config\\_t](#)  
*SDMA sw done configurations. [More...](#)*
- struct [sdma\\_p2p\\_config\\_t](#)  
*SDMA peripheral to peripheral R7 config. [More...](#)*
- struct [sdma\\_transfer\\_config\\_t](#)  
*SDMA transfer configuration. [More...](#)*
- struct [sdma\\_buffer\\_descriptor\\_t](#)  
*SDMA buffer descriptor structure. [More...](#)*
- struct [sdma\\_channel\\_control\\_t](#)  
*SDMA channel control descriptor structure. [More...](#)*
- struct [sdma\\_context\\_data\\_t](#)  
*SDMA context structure for each channel. [More...](#)*
- struct [sdma\\_handle\\_t](#)  
*SDMA transfer handle structure. [More...](#)*

## Typedefs

- typedef void(\* [sdma\\_callback](#) )(struct \_sdma\_handle \*handle, void \*userData, bool transferDone, uint32\_t bdIndex)  
*Define callback function for SDMA.*

## Enumerations

- enum `sdma_transfer_size_t` {  
`kSDMA_TransferSize1Bytes` = 0x1U,  
`kSDMA_TransferSize2Bytes` = 0x2U,  
`kSDMA_TransferSize3Bytes` = 0x3U,  
`kSDMA_TransferSize4Bytes` = 0x0U }  
*SDMA transfer configuration.*
- enum `sdma_bd_status_t` {  
`kSDMA_BDStatusDone` = 0x1U,  
`kSDMA_BDStatusWrap` = 0x2U,  
`kSDMA_BDStatusContinuous` = 0x4U,  
`kSDMA_BDStatusInterrupt` = 0x8U,  
`kSDMA_BDStatusError` = 0x10U,  
`kSDMA_BDStatusLast`,  
`kSDMA_BDStatusExtend` = 0x80U }  
*SDMA buffer descriptor status.*
- enum `sdma_bd_command_t` {  
`kSDMA_BDCommandSETDM` = 0x1U,  
`kSDMA_BDCommandGETDM` = 0x2U,  
`kSDMA_BDCommandSETPM` = 0x4U,  
`kSDMA_BDCommandGETPM` = 0x6U,  
`kSDMA_BDCommandSETCTX` = 0x7U,  
`kSDMA_BDCommandGETCTX` = 0x3U }  
*SDMA buffer descriptor command.*
- enum `sdma_context_switch_mode_t` {  
`kSDMA_ContextSwitchModeStatic` = 0x0U,  
`kSDMA_ContextSwitchModeDynamicLowPower`,  
`kSDMA_ContextSwitchModeDynamicWithNoLoop`,  
`kSDMA_ContextSwitchModeDynamic` }  
*SDMA context switch mode.*
- enum `sdma_clock_ratio_t` {  
`kSDMA_HalfARMClockFreq` = 0x0U,  
`kSDMA_ARMClockFreq` }  
*SDMA core clock frequency ratio to the ARM DMA interface.*
- enum `sdma_transfer_type_t` {  
`kSDMA_MemoryToMemory` = 0x0U,  
`kSDMA_PeripheralToMemory`,  
`kSDMA_MemoryToPeripheral`,  
`kSDMA_PeripheralToPeripheral` }  
*SDMA transfer type.*
- enum `sdma_peripheral_t` {

```

kSDMA_PeripheralTypeMemory = 0x0,
kSDMA_PeripheralTypeUART,
kSDMA_PeripheralTypeUART_SP,
kSDMA_PeripheralTypeSPDIF,
kSDMA_PeripheralNormal,
kSDMA_PeripheralNormal_SP,
kSDMA_PeripheralMultiFifoPDM,
kSDMA_PeripheralMultiFifoSaiRX,
kSDMA_PeripheralMultiFifoSaiTX,
kSDMA_PeripheralASRCM2P,
kSDMA_PeripheralASRCP2M,
kSDMA_PeripheralASRCP2P }

```

*Peripheral type use SDMA.*

- enum {
 

```

kStatus_SDMA_ERROR = MAKE_STATUS(kStatusGroup_SDMA, 0),
kStatus_SDMA_Busy = MAKE_STATUS(kStatusGroup_SDMA, 1) }

```

*\_sdma\_transfer\_status SDMA transfer status*
- enum {
 

```

kSDMA_MultiFifoWatermarkLevelMask = 0xFFFU,
kSDMA_MultiFifoNumsMask = 0xFU,
kSDMA_MultiFifoOffsetMask = 0xFU,
kSDMA_MultiFifoSwDoneMask = 0x1U,
kSDMA_MultiFifoSwDoneSelectorMask = 0xFU }

```

*\_sdma\_multi\_fifo\_mask SDMA multi fifo mask*
- enum {
 

```

kSDMA_MultiFifoWatermarkLevelShift = 0U,
kSDMA_MultiFifoNumsShift = 12U,
kSDMA_MultiFifoOffsetShift = 16U,
kSDMA_MultiFifoSwDoneShift = 23U,
kSDMA_MultiFifoSwDoneSelectorShift = 24U }

```

*\_sdma\_multi\_fifo\_shift SDMA multi fifo shift*
- enum {
 

```

kSDMA_DoneChannel0 = 0U,
kSDMA_DoneChannel1 = 1U,
kSDMA_DoneChannel2 = 2U,
kSDMA_DoneChannel3 = 3U,
kSDMA_DoneChannel4 = 4U,
kSDMA_DoneChannel5 = 5U,
kSDMA_DoneChannel6 = 6U,
kSDMA_DoneChannel7 = 7U }

```

*\_sdma\_done\_channel SDMA done channel*
- enum `sdma_done_src_t` {

```

kSDMA_DoneSrcSW = 0U,
kSDMA_DoneSrcHwEvent0U = 1U,
kSDMA_DoneSrcHwEvent1U = 2U,
kSDMA_DoneSrcHwEvent2U = 3U,
kSDMA_DoneSrcHwEvent3U = 4U,
kSDMA_DoneSrcHwEvent4U = 5U,
kSDMA_DoneSrcHwEvent5U = 6U,
kSDMA_DoneSrcHwEvent6U = 7U,
kSDMA_DoneSrcHwEvent7U = 8U,
kSDMA_DoneSrcHwEvent8U = 9U,
kSDMA_DoneSrcHwEvent9U = 10U,
kSDMA_DoneSrcHwEvent10U = 11U,
kSDMA_DoneSrcHwEvent11U = 12U,
kSDMA_DoneSrcHwEvent12U = 13U,
kSDMA_DoneSrcHwEvent13U = 14U,
kSDMA_DoneSrcHwEvent14U = 15U,
kSDMA_DoneSrcHwEvent15U = 16U,
kSDMA_DoneSrcHwEvent16U = 17U,
kSDMA_DoneSrcHwEvent17U = 18U,
kSDMA_DoneSrcHwEvent18U = 19U,
kSDMA_DoneSrcHwEvent19U = 20U,
kSDMA_DoneSrcHwEvent20U = 21U,
kSDMA_DoneSrcHwEvent21U = 22U,
kSDMA_DoneSrcHwEvent22U = 23U,
kSDMA_DoneSrcHwEvent23U = 24U,
kSDMA_DoneSrcHwEvent24U = 25U,
kSDMA_DoneSrcHwEvent25U = 26U,
kSDMA_DoneSrcHwEvent26U = 27U,
kSDMA_DoneSrcHwEvent27U = 28U,
kSDMA_DoneSrcHwEvent28U = 29U,
kSDMA_DoneSrcHwEvent29U = 30U,
kSDMA_DoneSrcHwEvent30U = 31U,
kSDMA_DoneSrcHwEvent31U = 32U }

```

*SDMA done source.*

## Driver version

- #define `FSL_SDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 1)`)  
*SDMA driver version.*

## SDMA initialization and de-initialization

- void `SDMA_Init` (`SDMAARM_Type *base`, const `sdma_config_t *config`)  
*Initializes the SDMA peripheral.*
- void `SDMA_Deinit` (`SDMAARM_Type *base`)  
*Deinitializes the SDMA peripheral.*

- void [SDMA\\_GetDefaultConfig](#) ([sdma\\_config\\_t](#) \*config)  
*Gets the SDMA default configuration structure.*
- void [SDMA\\_ResetModule](#) ([SDMAARM\\_Type](#) \*base)  
*Sets all SDMA core register to reset status.*

## SDMA Channel Operation

- static void [SDMA\\_EnableChannelErrorInterrupts](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Enables the interrupt source for the SDMA error.*
- static void [SDMA\\_DisableChannelErrorInterrupts](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Disables the interrupt source for the SDMA error.*

## SDMA Buffer Descriptor Operation

- void [SDMA\\_ConfigBufferDescriptor](#) ([sdma\\_buffer\\_descriptor\\_t](#) \*bd, [uint32\\_t](#) srcAddr, [uint32\\_t](#) destAddr, [sdma\\_transfer\\_size\\_t](#) busWidth, [size\\_t](#) bufferSize, bool isLast, bool enableInterrupt, bool isWrap, [sdma\\_transfer\\_type\\_t](#) type)  
*Sets buffer descriptor contents.*

## SDMA Channel Transfer Operation

- static void [SDMA\\_SetChannelPriority](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) channel, [uint8\\_t](#) priority)  
*Set SDMA channel priority.*
- static void [SDMA\\_SetSourceChannel](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) source, [uint32\\_t](#) channel-Mask)  
*Set SDMA request source mapping channel.*
- static void [SDMA\\_StartChannelSoftware](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Start a SDMA channel by software trigger.*
- static void [SDMA\\_StartChannelEvents](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Start a SDMA channel by hardware events.*
- static void [SDMA\\_StopChannel](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Stop a SDMA channel.*
- void [SDMA\\_SetContextSwitchMode](#) ([SDMAARM\\_Type](#) \*base, [sdma\\_context\\_switch\\_mode\\_t](#) mode)  
*Set the SDMA context switch mode.*

## SDMA Channel Status Operation

- static [uint32\\_t](#) [SDMA\\_GetChannelInterruptStatus](#) ([SDMAARM\\_Type](#) \*base)  
*Gets the SDMA interrupt status of all channels.*
- static void [SDMA\\_ClearChannelInterruptStatus](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Clear the SDMA channel interrupt status of specific channels.*
- static [uint32\\_t](#) [SDMA\\_GetChannelStopStatus](#) ([SDMAARM\\_Type](#) \*base)  
*Gets the SDMA stop status of all channels.*
- static void [SDMA\\_ClearChannelStopStatus](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) mask)  
*Clear the SDMA channel stop status of specific channels.*
- static [uint32\\_t](#) [SDMA\\_GetChannelPendStatus](#) ([SDMAARM\\_Type](#) \*base)  
*Gets the SDMA channel pending status of all channels.*
- static void [SDMA\\_ClearChannelPendStatus](#) ([SDMAARM\\_Type](#) \*base, [uint32\\_t](#) mask)

- *Clear the SDMA channel pending status of specific channels.*
- static uint32\_t [SDMA\\_GetErrorStatus](#) (SDMAARM\_Type \*base)  
*Gets the SDMA channel error status.*
- bool [SDMA\\_GetRequestSourceStatus](#) (SDMAARM\_Type \*base, uint32\_t source)  
*Gets the SDMA request source pending status.*

## SDMA Transactional Operation

- void [SDMA\\_CreateHandle](#) (sdma\_handle\_t \*handle, SDMAARM\_Type \*base, uint32\_t channel, sdma\_context\_data\_t \*context)  
*Creates the SDMA handle.*
- void [SDMA\\_InstallBDMemory](#) (sdma\_handle\_t \*handle, sdma\_buffer\_descriptor\_t \*BDPool, uint32\_t BDCount)  
*Installs the BDs memory pool into the SDMA handle.*
- void [SDMA\\_SetCallback](#) (sdma\_handle\_t \*handle, sdma\_callback callback, void \*userData)  
*Installs a callback function for the SDMA transfer.*
- void [SDMA\\_SetMultiFifoConfig](#) (sdma\_transfer\_config\_t \*config, uint32\_t fifoNums, uint32\_t fifoOffset)  
*multi fifo configurations.*
- void [SDMA\\_EnableSwDone](#) (SDMAARM\_Type \*base, sdma\_transfer\_config\_t \*config, uint8\_t sel, sdma\_peripheral\_t type)  
*enable sdma sw done feature.*
- void [SDMA\\_SetDoneConfig](#) (SDMAARM\_Type \*base, sdma\_transfer\_config\_t \*config, sdma\_peripheral\_t type, sdma\_done\_src\_t doneSrc)  
*sdma channel done configurations.*
- void [SDMA\\_LoadScript](#) (SDMAARM\_Type \*base, uint32\_t destAddr, void \*srcAddr, size\_t bufferSizeBytes)  
*load script to sdma program memory.*
- void [SDMA\\_DumpScript](#) (SDMAARM\_Type \*base, uint32\_t srcAddr, void \*destAddr, size\_t bufferSizeBytes)  
*dump script from sdma program memory.*
- static const char \* [SDMA\\_GetRamScriptVersion](#) (SDMAARM\_Type \*base)  
*Get RAM script version.*
- void [SDMA\\_PrepareTransfer](#) (sdma\_transfer\_config\_t \*config, uint32\_t srcAddr, uint32\_t destAddr, uint32\_t srcWidth, uint32\_t destWidth, uint32\_t bytesEachRequest, uint32\_t transferSize, uint32\_t eventSource, sdma\_peripheral\_t peripheral, sdma\_transfer\_type\_t type)  
*Prepares the SDMA transfer structure.*
- void [SDMA\\_PrepareP2PTransfer](#) (sdma\_transfer\_config\_t \*config, uint32\_t srcAddr, uint32\_t destAddr, uint32\_t srcWidth, uint32\_t destWidth, uint32\_t bytesEachRequest, uint32\_t transferSize, uint32\_t eventSource, uint32\_t eventSource1, sdma\_peripheral\_t peripheral, sdma\_p2p\_config\_t \*p2p)  
*Prepares the SDMA P2P transfer structure.*
- void [SDMA\\_SubmitTransfer](#) (sdma\_handle\_t \*handle, const sdma\_transfer\_config\_t \*config)  
*Submits the SDMA transfer request.*
- void [SDMA\\_StartTransfer](#) (sdma\_handle\_t \*handle)  
*SDMA starts transfer.*
- void [SDMA\\_StopTransfer](#) (sdma\_handle\_t \*handle)  
*SDMA stops transfer.*
- void [SDMA\\_AbortTransfer](#) (sdma\_handle\_t \*handle)  
*SDMA aborts transfer.*

- uint32\_t [SDMA\\_GetTransferredBytes](#) (sdma\_handle\_t \*handle)  
*Get transferred bytes while not using BD pools.*
- void [SDMA\\_HandleIRQ](#) (sdma\_handle\_t \*handle)  
*SDMA IRQ handler for complete a buffer descriptor transfer.*

## 20.3 Data Structure Documentation

### 20.3.1 struct sdma\_config\_t

#### Data Fields

- bool [enableRealTimeDebugPin](#)  
*If enable real-time debug pin, default is closed to reduce power consumption.*
- bool [isSoftwareResetClearLock](#)  
*If software reset clears the LOCK bit which prevent writing SDMA scripts into SDMA.*
- [sdma\\_clock\\_ratio\\_t](#) ratio  
*SDMA core clock ratio to ARM platform DMA interface.*

#### Field Documentation

(1) bool sdma\_config\_t::enableRealTimeDebugPin

(2) bool sdma\_config\_t::isSoftwareResetClearLock

### 20.3.2 struct sdma\_multi\_fifo\_config\_t

#### Data Fields

- uint8\_t [fifoNums](#)  
*fifo numbers*
- uint8\_t [fifoOffset](#)  
*offset between multi fifo data register address*

### 20.3.3 struct sdma\_sw\_done\_config\_t

#### Data Fields

- bool [enableSwDone](#)  
*true is enable sw done, false is disable*
- uint8\_t [swDoneSel](#)  
*sw done channel number per peripheral type*



### 20.3.4 struct sdma\_p2p\_config\_t

#### Data Fields

- uint8\_t [sourceWatermark](#)  
*lower watermark value*
- uint8\_t [destWatermark](#)  
*higher watermark value*
- bool [continuousTransfer](#)  
*0: the amount of samples to be transferred is equal to the cont field of mode word 1: the amount of samples to be transferred is unknown and script will keep on transferring as long as both events are detected and script must be stopped by application.*

#### Field Documentation

(1) bool sdma\_p2p\_config\_t::continuousTransfer

### 20.3.5 struct sdma\_transfer\_config\_t

This structure configures the source/destination transfer attribute.

#### Data Fields

- uint32\_t [srcAddr](#)  
*Source address of the transfer.*
- uint32\_t [destAddr](#)  
*Destination address of the transfer.*
- [sdma\\_transfer\\_size\\_t](#) [srcTransferSize](#)  
*Source data transfer size.*
- [sdma\\_transfer\\_size\\_t](#) [destTransferSize](#)  
*Destination data transfer size.*
- uint32\_t [bytesPerRequest](#)  
*Bytes to transfer in a minor loop.*
- uint32\_t [transferSzie](#)  
*Bytes to transfer for this descriptor.*
- uint32\_t [scriptAddr](#)  
*SDMA script address located in SDMA ROM.*
- uint32\_t [eventSource](#)  
*Event source number for the channel.*
- uint32\_t [eventSource1](#)  
*event source 1*
- bool [isEventIgnore](#)  
*True means software trigger, false means hardware trigger.*
- bool [isSoftTriggerIgnore](#)  
*If ignore the HE bit, 1 means use hardware events trigger, 0 means software trigger.*
- [sdma\\_transfer\\_type\\_t](#) [type](#)  
*Transfer type, transfer type used to decide the SDMA script.*
- [sdma\\_multi\\_fifo\\_config\\_t](#) [multiFifo](#)



- *multi fifo configurations*  
• `sdma_sw_done_config_t` `swDone`  
    *sw done selector*
- `uint32_t` `watermarkLevel`  
    *watermark level*
- `uint32_t` `eventMask0`  
    *event mask 0*
- `uint32_t` `eventMask1`  
    *event mask 1*

### Field Documentation

- (1) `sdma_transfer_size_t` `sdma_transfer_config_t::srcTransferSize`
- (2) `sdma_transfer_size_t` `sdma_transfer_config_t::destTransferSize`
- (3) `uint32_t` `sdma_transfer_config_t::scriptAddr`
- (4) `uint32_t` `sdma_transfer_config_t::eventSource`

0 means no event, use software trigger

- (5) `sdma_transfer_type_t` `sdma_transfer_config_t::type`

### 20.3.6 struct `sdma_buffer_descriptor_t`

This structure is a buffer descriptor, this structure describes the buffer start address and other options

### Data Fields

- `uint32_t` `count`: 16  
    *Bytes of the buffer length for this buffer descriptor.*
- `uint32_t` `status`: 8  
    *E,R,I,C,W,D status bits stored here.*
- `uint32_t` `command`: 8  
    *command mostly used for channel 0*
- `uint32_t` `bufferAddr`  
    *Buffer start address for this descriptor.*
- `uint32_t` `extendBufferAddr`  
    *External buffer start address, this is an optional for a transfer.*

## Field Documentation

- (1) `uint32_t sdma_buffer_descriptor_t::count`
- (2) `uint32_t sdma_buffer_descriptor_t::bufferAddr`
- (3) `uint32_t sdma_buffer_descriptor_t::extendBufferAddr`

20.3.7 `struct sdma_channel_control_t`

## Data Fields

- `uint32_t currentBDAddr`  
*Address of current buffer descriptor processed.*
- `uint32_t baseBDAddr`  
*The start address of the buffer descriptor array.*
- `uint32_t channelDesc`  
*Optional for transfer.*
- `uint32_t status`  
*Channel status.*

20.3.8 `struct sdma_context_data_t`

This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

## Data Fields

- `uint32_t GeneralReg [8]`  
*8 general regsiters used for SDMA RISC core*

20.3.9 `struct sdma_handle_t`

## Data Fields

- `sdma_callback callback`  
*Callback function for major count exhausted.*
- `void * userData`  
*Callback function parameter.*
- `SDMAARM_Type * base`  
*SDMA peripheral base address.*
- `sdma_buffer_descriptor_t * BDPool`  
*Pointer to memory stored BD arrays.*
- `uint32_t bdCount`  
*How many buffer descriptor.*
- `uint32_t bdIndex`

- *How many buffer descriptor.*  
uint32\_t [eventSource](#)
- *Event source count for the channel.*  
uint32\_t [eventSource1](#)
- *Event source 1 count for the channel.*  
[sdma\\_context\\_data\\_t](#) \* [context](#)
- *Channel context to execute in SDMA.*  
uint8\_t [channel](#)
- *SDMA channel number.*  
uint8\_t [priority](#)
- *SDMA channel priority.*  
uint8\_t [flags](#)
- *The status of the current channel.*  
bool [isRamscriptLoaded](#)
- *Flag to indicate the status of ram script.*

### Field Documentation

- (1) `sdma_callback sdma_handle_t::callback`
- (2) `void* sdma_handle_t::userData`
- (3) `SDMAARM_Type* sdma_handle_t::base`
- (4) `sdma_buffer_descriptor_t* sdma_handle_t::BDPool`
- (5) `uint8_t sdma_handle_t::channel`
- (6) `uint8_t sdma_handle_t::flags`

## 20.4 Macro Definition Documentation

### 20.4.1 #define FSL\_SDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 1))

Version 2.4.1.

## 20.5 Typedef Documentation

### 20.5.1 typedef void(\* sdma\_callback)(struct \_sdma\_handle \*handle, void \*userData, bool transferDone, uint32\_t bdIndex)

## 20.6 Enumeration Type Documentation

### 20.6.1 enum sdma\_transfer\_size\_t

Enumerator

- kSDMA\_TransferSize1Bytes* Source/Destination data transfer size is 1 byte every time.
- kSDMA\_TransferSize2Bytes* Source/Destination data transfer size is 2 bytes every time.
- kSDMA\_TransferSize3Bytes* Source/Destination data transfer size is 3 bytes every time.

***kSDMA\_TransferSize4Bytes*** Source/Destination data transfer size is 4 bytes every time.

## 20.6.2 enum sdma\_bd\_status\_t

Enumerator

***kSDMA\_BDStatusDone*** BD ownership, 0 means ARM core owns the BD, while 1 means SDMA owns BD.

***kSDMA\_BDStatusWrap*** While this BD is last one, the next BD will be the first one.

***kSDMA\_BDStatusContinuous*** Buffer is allowed to transfer/receive to/from multiple buffers.

***kSDMA\_BDStatusInterrupt*** While this BD finished, send an interrupt.

***kSDMA\_BDStatusError*** Error occurred on buffer descriptor command.

***kSDMA\_BDStatusLast*** This BD is the last BD in this array. It means the transfer ended after this buffer

***kSDMA\_BDStatusExtend*** Buffer descriptor extend status for SDMA scripts.

## 20.6.3 enum sdma\_bd\_command\_t

Enumerator

***kSDMA\_BDCommandSETDM*** Load SDMA data memory from ARM core memory buffer.

***kSDMA\_BDCommandGETDM*** Copy SDMA data memory to ARM core memory buffer.

***kSDMA\_BDCommandSETPM*** Load SDMA program memory from ARM core memory buffer.

***kSDMA\_BDCommandGETPM*** Copy SDMA program memory to ARM core memory buffer.

***kSDMA\_BDCommandSETCTX*** Load context for one channel into SDMA RAM from ARM platform memory buffer.

***kSDMA\_BDCommandGETCTX*** Copy context for one channel from SDMA RAM to ARM platform memory buffer.

## 20.6.4 enum sdma\_context\_switch\_mode\_t

Enumerator

***kSDMA\_ContextSwitchModeStatic*** SDMA context switch mode static.

***kSDMA\_ContextSwitchModeDynamicLowPower*** SDMA context switch mode dynamic with low power.

***kSDMA\_ContextSwitchModeDynamicWithNoLoop*** SDMA context switch mode dynamic with no loop.

***kSDMA\_ContextSwitchModeDynamic*** SDMA context switch mode dynamic.

### 20.6.5 enum sdma\_clock\_ratio\_t

Enumerator

*kSDMA\_HalfARMClockFreq* SDMA core clock frequency half of ARM platform.

*kSDMA\_ARMClockFreq* SDMA core clock frequency equals to ARM platform.

### 20.6.6 enum sdma\_transfer\_type\_t

Enumerator

*kSDMA\_MemoryToMemory* Transfer from memory to memory.

*kSDMA\_PeripheralToMemory* Transfer from peripheral to memory.

*kSDMA\_MemoryToPeripheral* Transfer from memory to peripheral.

*kSDMA\_PeripheralToPeripheral* Transfer from peripheral to peripheral.

### 20.6.7 enum sdma\_peripheral\_t

Enumerator

*kSDMA\_PeripheralTypeMemory* Peripheral DDR memory.

*kSDMA\_PeripheralTypeUART* UART use SDMA.

*kSDMA\_PeripheralTypeUART\_SP* UART instance in SPBA use SDMA.

*kSDMA\_PeripheralTypeSPDIF* SPDIF use SDMA.

*kSDMA\_PeripheralNormal* Normal peripheral use SDMA.

*kSDMA\_PeripheralNormal\_SP* Normal peripheral in SPBA use SDMA.

*kSDMA\_PeripheralMultiFifoPDM* multi fifo PDM

*kSDMA\_PeripheralMultiFifoSaiRX* multi fifo sai rx use SDMA

*kSDMA\_PeripheralMultiFifoSaiTX* multi fifo sai tx use SDMA

*kSDMA\_PeripheralASRCM2P* asrc m2p

*kSDMA\_PeripheralASRCP2M* asrc p2m

*kSDMA\_PeripheralASRCP2P* asrc p2p

### 20.6.8 anonymous enum

Enumerator

*kStatus\_SDMA\_ERROR* SDMA context error.

*kStatus\_SDMA\_Busy* Channel is busy and can't handle the transfer request.

### 20.6.9 anonymous enum

Enumerator

*kSDMA\_MultiFifoWatermarkLevelMask* multi fifo watermark level mask  
*kSDMA\_MultiFifoNumsMask* multi fifo nums mask  
*kSDMA\_MultiFifoOffsetMask* multi fifo offset mask  
*kSDMA\_MultiFifoSwDoneMask* multi fifo sw done mask  
*kSDMA\_MultiFifoSwDoneSelectorMask* multi fifo sw done selector mask

### 20.6.10 anonymous enum

Enumerator

*kSDMA\_MultiFifoWatermarkLevelShift* multi fifo watermark level shift  
*kSDMA\_MultiFifoNumsShift* multi fifo nums shift  
*kSDMA\_MultiFifoOffsetShift* multi fifo offset shift  
*kSDMA\_MultiFifoSwDoneShift* multi fifo sw done shift  
*kSDMA\_MultiFifoSwDoneSelectorShift* multi fifo sw done selector shift

### 20.6.11 anonymous enum

Enumerator

*kSDMA\_DoneChannel0* SDMA done channel 0.  
*kSDMA\_DoneChannel1* SDMA done channel 1.  
*kSDMA\_DoneChannel2* SDMA done channel 2.  
*kSDMA\_DoneChannel3* SDMA done channel 3.  
*kSDMA\_DoneChannel4* SDMA done channel 4.  
*kSDMA\_DoneChannel5* SDMA done channel 5.  
*kSDMA\_DoneChannel6* SDMA done channel 6.  
*kSDMA\_DoneChannel7* SDMA done channel 7.

### 20.6.12 enum sdma\_done\_src\_t

Enumerator

*kSDMA\_DoneSrcSW* software done  
*kSDMA\_DoneSrcHwEvent0U* HW event 0 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent1U* HW event 1 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent2U* HW event 2 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent3U* HW event 3 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent4U* HW event 4 is used for DONE event.

*kSDMA\_DoneSrcHwEvent5U* HW event 5 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent6U* HW event 6 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent7U* HW event 7 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent8U* HW event 8 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent9U* HW event 9 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent10U* HW event 10 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent11U* HW event 11 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent12U* HW event 12 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent13U* HW event 13 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent14U* HW event 14 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent15U* HW event 15 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent16U* HW event 16 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent17U* HW event 17 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent18U* HW event 18 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent19U* HW event 19 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent20U* HW event 20 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent21U* HW event 21 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent22U* HW event 22 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent23U* HW event 23 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent24U* HW event 24 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent25U* HW event 25 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent26U* HW event 26 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent27U* HW event 27 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent28U* HW event 28 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent29U* HW event 29 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent30U* HW event 30 is used for DONE event.  
*kSDMA\_DoneSrcHwEvent31U* HW event 31 is used for DONE event.

## 20.7 Function Documentation

### 20.7.1 void SDMA\_Init ( SDMAARM\_Type \* *base*, const sdma\_config\_t \* *config* )

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

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>base</i>   | SDMA peripheral base address.                                  |
| <i>config</i> | A pointer to the configuration structure, see "sdma_config_t". |

Note

This function enables the minor loop map feature.

### 20.7.2 void SDMA\_Deinit ( SDMAARM\_Type \* *base* )

This function gates the SDMA clock.



## Parameters

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

**20.7.3 void SDMA\_GetDefaultConfig ( sdma\_config\_t \* *config* )**

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

```
* config.enableRealTimeDebugPin = false;
* config.isSoftwareResetClearLock = true;
* config.ratio = kSDMA_HalfARMClockFreq;
*
```

## Parameters

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

**20.7.4 void SDMA\_ResetModule ( SDMAARM\_Type \* *base* )**

If only reset ARM core, SDMA register cannot return to reset value, shall call this function to reset all SDMA register to reset value. But the internal status cannot be reset.

## Parameters

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

**20.7.5 static void SDMA\_EnableChannelErrorInterrupts ( SDMAARM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Enable this will trigger an interrupt while SDMA occurs error while executing scripts.

## Parameters

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

**20.7.6 static void SDMA\_DisableChannelErrorInterrupts ( SDMAARM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

## Parameters

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

**20.7.7 void SDMA\_ConfigBufferDescriptor ( sdma\_buffer\_descriptor\_t \* *bd*, uint32\_t *srcAddr*, uint32\_t *destAddr*, sdma\_transfer\_size\_t *busWidth*, size\_t *bufferSize*, bool *isLast*, bool *enableInterrupt*, bool *isWrap*, sdma\_transfer\_type\_t *type* )**

This function sets the descriptor contents such as source, dest address and status bits.

## Parameters

|                        |                                                                                                                                                    |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>bd</i>              | Pointer to the buffer descriptor structure.                                                                                                        |
| <i>srcAddr</i>         | Source address for the buffer descriptor.                                                                                                          |
| <i>destAddr</i>        | Destination address for the buffer descriptor.                                                                                                     |
| <i>busWidth</i>        | The transfer width, it only can be a member of sdma_transfer_size_t.                                                                               |
| <i>bufferSize</i>      | Buffer size for this descriptor, this number shall less than 0xFFFF. If need to transfer a big size, shall divide into several buffer descriptors. |
| <i>isLast</i>          | Is the buffer descriptor the last one for the channel to transfer. If only one descriptor used for the channel, this bit shall set to TRUE.        |
| <i>enableInterrupt</i> | If trigger an interrupt while this buffer descriptor transfer finished.                                                                            |
| <i>isWrap</i>          | Is the buffer descriptor need to be wrapped. While this bit set to true, it will automatically wrap to the first buffer descriptor to do transfer. |
| <i>type</i>            | Transfer type, memory to memory, peripheral to memory or memory to peripheral.                                                                     |

**20.7.8 static void SDMA\_SetChannelPriority ( SDMAARM\_Type \* *base*, uint32\_t *channel*, uint8\_t *priority* ) [inline], [static]**

This function sets the channel priority. The default value is 0 for all channels, priority 0 will prevents channel from starting, so the priority must be set before start a channel.

## Parameters

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

#### 20.7.9 static void SDMA\_SetSourceChannel ( SDMAARM\_Type \* *base*, uint32\_t *source*, uint32\_t *channelMask* ) [inline], [static]

This function sets which channel will be triggered by the dma request source.

Parameters

|                    |                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | SDMA peripheral base address.                                                                                                                                                       |
| <i>source</i>      | SDMA dma request source number.                                                                                                                                                     |
| <i>channelMask</i> | SDMA channel mask. 1 means channel 0, 2 means channel 1, 4 means channel 3. SDMA supports an event trigger multi-channel. A channel can also be triggered by several source events. |

#### 20.7.10 static void SDMA\_StartChannelSoftware ( SDMAARM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function start a channel.

Parameters

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

#### 20.7.11 static void SDMA\_StartChannelEvents ( SDMAARM\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function start a channel.

Parameters

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

**20.7.12** `static void SDMA_StopChannel ( SDMAARM_Type * base, uint32_t channel ) [inline], [static]`

This function stops a channel.

## Parameters

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

**20.7.13 void SDMA\_SetContextSwitchMode ( SDMAARM\_Type \* *base*,  
sdma\_context\_switch\_mode\_t *mode* )**

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | SDMA peripheral base address. |
| <i>mode</i> | SDMA context switch mode.     |

**20.7.14 static uint32\_t SDMA\_GetChannelInterruptStatus ( SDMAARM\_Type \* *base*  
) [inline], [static]**

## Parameters

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

## Returns

The interrupt status for all channels. Check the relevant bits for specific channel.

**20.7.15 static void SDMA\_ClearChannelInterruptStatus ( SDMAARM\_Type \* *base*,  
uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | SDMA peripheral base address.            |
| <i>mask</i> | The interrupt status need to be cleared. |

**20.7.16 static uint32\_t SDMA\_GetChannelStopStatus ( SDMAARM\_Type \* *base* )  
[inline], [static]**

## Parameters

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

## Returns

The stop status for all channels. Check the relevant bits for specific channel.

**20.7.17** `static void SDMA_ClearChannelStopStatus ( SDMAARM_Type * base,  
uint32_t mask ) [inline], [static]`

## Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | SDMA peripheral base address.       |
| <i>mask</i> | The stop status need to be cleared. |

**20.7.18** `static uint32_t SDMA_GetChannelPendStatus ( SDMAARM_Type * base )  
[inline], [static]`

## Parameters

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

## Returns

The pending status for all channels. Check the relevant bits for specific channel.

**20.7.19** `static void SDMA_ClearChannelPendStatus ( SDMAARM_Type * base,  
uint32_t mask ) [inline], [static]`

## Parameters

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

|             |                                        |
|-------------|----------------------------------------|
| <i>mask</i> | The pending status need to be cleared. |
|-------------|----------------------------------------|

### 20.7.20 **static uint32\_t SDMA\_GetErrorStatus ( SDMAARM\_Type \* *base* )** **[inline], [static]**

SDMA channel error flag is asserted while an incoming DMA request was detected and it triggers a channel that is already pending or being serviced. This probably means there is an overflow of data for that channel.

Parameters

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

Returns

The error status for all channels. Check the relevant bits for specific channel.

### 20.7.21 **bool SDMA\_GetRequestSourceStatus ( SDMAARM\_Type \* *base*, uint32\_t *source* )**

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | SDMA peripheral base address. |
| <i>source</i> | DMA request source number.    |

Returns

True means the request source is pending, otherwise not pending.

### 20.7.22 **void SDMA\_CreateHandle ( sdma\_handle\_t \* *handle*, SDMAARM\_Type \* *base*, uint32\_t *channel*, sdma\_context\_data\_t \* *context* )**

This function is called if using the transactional API for SDMA. This function initializes the internal state of the SDMA handle.

## Parameters

|                |                                                                                                                                                                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | SDMA handle pointer. The SDMA handle stores callback function and parameters.                                                                                                                                                                                               |
| <i>base</i>    | SDMA peripheral base address.                                                                                                                                                                                                                                               |
| <i>channel</i> | SDMA channel number.                                                                                                                                                                                                                                                        |
| <i>context</i> | Context structure for the channel to download into SDMA. Users shall make sure the context located in a non-cacheable memory, or it will cause SDMA run fail. Users shall not touch the context contents, it only be filled by SDMA driver in SDMA_SubmitTransfer function. |

### 20.7.23 void SDMA\_InstallBDMemory ( sdma\_handle\_t \* *handle*, sdma\_buffer\_descriptor\_t \* *BDPool*, uint32\_t *BDCount* )

This function is called after the SDMA\_CreateHandle to use multi-buffer feature.

## Parameters

|                |                                                                          |
|----------------|--------------------------------------------------------------------------|
| <i>handle</i>  | SDMA handle pointer.                                                     |
| <i>BDPool</i>  | A memory pool to store BDs. It must be located in non-cacheable address. |
| <i>BDCount</i> | The number of BD slots.                                                  |

### 20.7.24 void SDMA\_SetCallback ( sdma\_handle\_t \* *handle*, sdma\_callback *callback*, void \* *userData* )

This callback is called in the SDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>handle</i>   | SDMA handle pointer.                   |
| <i>callback</i> | SDMA callback function pointer.        |
| <i>userData</i> | A parameter for the callback function. |

### 20.7.25 void SDMA\_SetMultiFifoConfig ( sdma\_transfer\_config\_t \* *config*, uint32\_t *fifoNums*, uint32\_t *fifoOffset* )

This api is used to support multi fifo for SDMA, if user want to get multi fifo data, then this api should be called before submit transfer.



## Parameters

|                   |                                                                                |
|-------------------|--------------------------------------------------------------------------------|
| <i>config</i>     | transfer configurations.                                                       |
| <i>fifoNums</i>   | fifo numbers that multi fifo operation perform, support up to 15 fifo numbers. |
| <i>fifoOffset</i> | fifoOffset = fifo address offset / sizeof(uint32_t) - 1.                       |

**20.7.26 void SDMA\_EnableSwDone ( SDMAARM\_Type \* *base*, sdma\_transfer\_config\_t \* *config*, uint8\_t *sel*, sdma\_peripheral\_t *type* )**

**Deprecated** Do not use this function. It has been superceded by [SDMA\\_SetDoneConfig](#).

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | SDMA base.                                                                              |
| <i>config</i> | transfer configurations.                                                                |
| <i>sel</i>    | sw done selector.                                                                       |
| <i>type</i>   | peripheral type is used to determine the corresponding peripheral sw done selector bit. |

**20.7.27 void SDMA\_SetDoneConfig ( SDMAARM\_Type \* *base*, sdma\_transfer\_config\_t \* *config*, sdma\_peripheral\_t *type*, sdma\_done\_src\_t *doneSrc* )**

## Parameters

|                |                            |
|----------------|----------------------------|
| <i>base</i>    | SDMA base.                 |
| <i>config</i>  | transfer configurations.   |
| <i>type</i>    | peripheral type.           |
| <i>doneSrc</i> | reference sdma_done_src_t. |

**20.7.28 void SDMA\_LoadScript ( SDMAARM\_Type \* *base*, uint32\_t *destAddr*, void \* *srcAddr*, size\_t *bufferSizeBytes* )**

## Parameters

|                        |                                                             |
|------------------------|-------------------------------------------------------------|
| <i>base</i>            | SDMA base.                                                  |
| <i>destAddr</i>        | dest script address, should be SDMA program memory address. |
| <i>srcAddr</i>         | source address of target script.                            |
| <i>bufferSizeBytes</i> | bytes size of script.                                       |

**20.7.29 void SDMA\_DumpScript ( SDMAARM\_Type \* *base*, uint32\_t *srcAddr*, void \* *destAddr*, size\_t *bufferSizeBytes* )**

## Parameters

|                        |                                        |
|------------------------|----------------------------------------|
| <i>base</i>            | SDMA base.                             |
| <i>srcAddr</i>         | should be SDMA program memory address. |
| <i>destAddr</i>        | address to store scripts.              |
| <i>bufferSizeBytes</i> | bytes size of script.                  |

**20.7.30 static const char\* SDMA\_GetRamScriptVersion ( SDMAARM\_Type \* *base* ) [inline], [static]**

## Parameters

|             |            |
|-------------|------------|
| <i>base</i> | SDMA base. |
|-------------|------------|

## Returns

The script version of RAM.

**20.7.31 void SDMA\_PrepareTransfer ( sdma\_transfer\_config\_t \* *config*, uint32\_t *srcAddr*, uint32\_t *destAddr*, uint32\_t *srcWidth*, uint32\_t *destWidth*, uint32\_t *bytesEachRequest*, uint32\_t *transferSize*, uint32\_t *eventSource*, sdma\_peripheral\_t *peripheral*, sdma\_transfer\_type\_t *type* )**

This function prepares the transfer configuration structure according to the user input.

## Parameters

|                         |                                                                                 |
|-------------------------|---------------------------------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type <code>sdma_transfer_t</code> .         |
| <i>srcAddr</i>          | SDMA transfer source address.                                                   |
| <i>destAddr</i>         | SDMA transfer destination address.                                              |
| <i>srcWidth</i>         | SDMA transfer source address width(bytes).                                      |
| <i>destWidth</i>        | SDMA transfer destination address width(bytes).                                 |
| <i>bytesEachRequest</i> | SDMA transfer bytes per channel request.                                        |
| <i>transferSize</i>     | SDMA transfer bytes to be transferred.                                          |
| <i>eventSource</i>      | Event source number for the transfer, if use software trigger, just write 0.    |
| <i>peripheral</i>       | Peripheral type, used to decide if need to use some special scripts.            |
| <i>type</i>             | SDMA transfer type. Used to decide the correct SDMA script address in SDMA ROM. |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error.

**20.7.32** `void SDMA_PrepareP2PTransfer ( sdma_transfer_config_t * config, uint32_t srcAddr, uint32_t destAddr, uint32_t srcWidth, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferSize, uint32_t eventSource, uint32_t eventSource1, sdma_peripheral_t peripheral, sdma_p2p_config_t * p2p )`

This function prepares the transfer configuration structure according to the user input.

## Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>config</i>   | The user configuration structure of type <code>sdma_transfer_t</code> . |
| <i>srcAddr</i>  | SDMA transfer source address.                                           |
| <i>destAddr</i> | SDMA transfer destination address.                                      |

|                         |                                                                      |
|-------------------------|----------------------------------------------------------------------|
| <i>srcWidth</i>         | SDMA transfer source address width(bytes).                           |
| <i>destWidth</i>        | SDMA transfer destination address width(bytes).                      |
| <i>bytesEachRequest</i> | SDMA transfer bytes per channel request.                             |
| <i>transferSize</i>     | SDMA transfer bytes to be transferred.                               |
| <i>eventSource</i>      | Event source number for the transfer.                                |
| <i>eventSource1</i>     | Event source1 number for the transfer.                               |
| <i>peripheral</i>       | Peripheral type, used to decide if need to use some special scripts. |
| <i>p2p</i>              | sdma p2p configuration pointer.                                      |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error.

### 20.7.33 void SDMA\_SubmitTransfer ( sdma\_handle\_t \* *handle*, const sdma\_transfer\_config\_t \* *config* )

This function submits the SDMA transfer request according to the transfer configuration structure.

## Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>handle</i> | SDMA handle pointer.                              |
| <i>config</i> | Pointer to SDMA transfer configuration structure. |

### 20.7.34 void SDMA\_StartTransfer ( sdma\_handle\_t \* *handle* )

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | SDMA handle pointer. |
|---------------|----------------------|

### 20.7.35 void SDMA\_StopTransfer ( sdma\_handle\_t \* *handle* )

This function disables the channel request to pause the transfer. Users can call [SDMA\\_StartTransfer\(\)](#) again to resume the transfer.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | SDMA handle pointer. |
|---------------|----------------------|

**20.7.36 void SDMA\_AbortTransfer ( sdma\_handle\_t \* *handle* )**

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

## Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

**20.7.37 uint32\_t SDMA\_GetTransferredBytes ( sdma\_handle\_t \* *handle* )**

This function returns the buffer descriptor count value if not using buffer descriptor. While do a simple transfer, which only uses one descriptor, the SDMA driver inside handle the buffer descriptor. In uart receive case, it can tell users how many data already received, also it can tells users how many data transfferd while error occurred. Notice, the count would not change while transfer is on-going using default SDMA script.

## Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

## Returns

Transferred bytes.

**20.7.38 void SDMA\_HandleIRQ ( sdma\_handle\_t \* *handle* )**

This function clears the interrupt flags and also handle the CCB for the channel.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | SDMA handle pointer. |
|---------------|----------------------|

## Chapter 21

# SEMA4: Hardware Semaphores Driver

### 21.1 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) (((volatile uint8\_t \*)(&((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.*

## Driver version

- #define **FSL\_SEMA4\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 0, 3))  
*SEMA4 driver version.*

## 21.2 Macro Definition Documentation

### 21.2.1 #define SEMA4\_GATE\_NUM\_RESET\_ALL (64U)

## 21.3 Function Documentation

### 21.3.1 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. |
|-------------|--------------------------------|

### 21.3.2 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. |
|-------------|--------------------------------|

### 21.3.3 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. |

#### 21.3.4 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.      |

#### 21.3.5 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.         |

#### 21.3.6 static int32\_t SEMA4\_GetLockProc ( SEMA4\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function checks the lock status of a specific SEMA4 gate.



## 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.

### 21.3.7 **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. |

### 21.3.8 **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. |
|---------------------|--------------------------------------|

### 21.3.9 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. |

### 21.3.10 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. |

### 21.3.11 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.

## 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.

### 21.3.12 `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. |

### 21.3.13 `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 22

# TMU: Thermal Management Unit Driver

### 22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the thermal management unit (TMU) module of MCUXpresso SDK devices.

### 22.2 Typical use case

#### 22.2.1 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, 1, 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_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.*

- enum `tmu_amplifier_gain_t` {  
    `kTMU_AmplifierGain6_34` = 0U,  
    `kTMU_AmplifierGain6_485` = 1U,  
    `kTMU_AmplifierGain6_63` = 2U,  
    `kTMU_AmplifierGain6_775` = 3U,  
    `kTMU_AmplifierGain6_92` = 4U,  
    `kTMU_AmplifierGain7_065` = 5U,  
    `kTMU_AmplifierGain7_21` = 6U,  
    `kTMU_AmplifierGain7_355` = 7U,  
    `kTMU_AmplifierGain7_5` = 8U,  
    `kTMU_AmplifierGain7_645` = 9U,  
    `kTMU_AmplifierGain7_79` = 10U,  
    `kTMU_AmplifierGain7_935` = 11U,  
    `kTMU_AmplifierGain8_08` = 12U,  
    `kTMU_AmplifierGain8_225` = 13U,  
    `kTMU_AmplifierGain8_37` = 14U,  
    `kTMU_AmplifierGain8_515` = 15U }

*Amplifier gain setting.*

- enum `tmu_amplifier_reference_voltage_t` {

```

kTMU_AmplifierReferenceVoltage510 = 0U,
kTMU_AmplifierReferenceVoltage517_5 = 1U,
kTMU_AmplifierReferenceVoltage525 = 2U,
kTMU_AmplifierReferenceVoltage532_5 = 3U,
kTMU_AmplifierReferenceVoltage540 = 4U,
kTMU_AmplifierReferenceVoltage547_5 = 5U,
kTMU_AmplifierReferenceVoltage555 = 6U,
kTMU_AmplifierReferenceVoltage562_5 = 7U,
kTMU_AmplifierReferenceVoltage570 = 8U,
kTMU_AmplifierReferenceVoltage577_5 = 9U,
kTMU_AmplifierReferenceVoltage585 = 10U,
kTMU_AmplifierReferenceVoltage592_5 = 11U,
kTMU_AmplifierReferenceVoltage600 = 12U,
kTMU_AmplifierReferenceVoltage607_5 = 13U,
kTMU_AmplifierReferenceVoltage615 = 14U,
kTMU_AmplifierReferenceVoltage622_5 = 15U,
kTMU_AmplifierReferenceVoltage630 = 16U,
kTMU_AmplifierReferenceVoltage637_5 = 17U,
kTMU_AmplifierReferenceVoltage645 = 18U,
kTMU_AmplifierReferenceVoltage652_5 = 19U,
kTMU_AmplifierReferenceVoltage660 = 20U,
kTMU_AmplifierReferenceVoltage667_5 = 21U,
kTMU_AmplifierReferenceVoltage675 = 22U,
kTMU_AmplifierReferenceVoltage682_5 = 23U,
kTMU_AmplifierReferenceVoltage690 = 24U,
kTMU_AmplifierReferenceVoltage697_5 = 25U,
kTMU_AmplifierReferenceVoltage705 = 26U,
kTMU_AmplifierReferenceVoltage712_5 = 27U,
kTMU_AmplifierReferenceVoltage720 = 28U,
kTMU_AmplifierReferenceVoltage727_5 = 29U,
kTMU_AmplifierReferenceVoltage735 = 30U,
kTMU_AmplifierReferenceVoltage742_5 = 31U }

```

*Amplifier reference voltage 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 monitoring the temperature sensor.*
- 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.*
- [status\\_t](#) [TMU\\_GetImmediateTemperature](#) (TMU\_Type \*base, uint32\_t \*temperature)  
*Get the last immediate temperature at site.*
- [status\\_t](#) [TMU\\_GetAverageTemperature](#) (TMU\_Type \*base, uint32\_t \*temperature)  
*Get the last average temperature at site.*
- void [TMU\\_SetHighTemperatureThreshold](#) (TMU\_Type \*base, const [tmu\\_threshold\\_config\\_t](#) \*config)  
*Configure the high temperature threshold value and enable/disable relevant threshold.*

## 22.3 Data Structure Documentation

### 22.3.1 struct tmu\_threshold\_config\_t

#### Data Fields

- bool [immediateThresholdEnable](#)  
*Enable high temperature immediate threshold.*
- bool [AverageThresholdEnable](#)  
*Enable high temperature average threshold.*
- bool [AverageCriticalThresholdEnable](#)  
*Enable high temperature average critical threshold.*
- uint8\_t [immediateThresholdValue](#)  
*Range: 10U-125U.*
- uint8\_t [averageThresholdValue](#)  
*Range: 10U-125U.*
- uint8\_t [averageCriticalThresholdValue](#)  
*Range: 10U-125U.*

#### Field Documentation

- (1) **bool tmu\_threshold\_config\_t::immediateThresholdEnable**
- (2) **bool tmu\_threshold\_config\_t::AverageThresholdEnable**
- (3) **bool tmu\_threshold\_config\_t::AverageCriticalThresholdEnable**
- (4) **uint8\_t tmu\_threshold\_config\_t::immediateThresholdValue**

Valid when corresponding threshold is enabled. High temperature immediate threshold value. Determines the current upper temperature threshold, for any enabled monitored site.

- (5) **uint8\_t tmu\_threshold\_config\_t::averageThresholdValue**

Valid when corresponding threshold is enabled. High temperature average threshold value. Determines the average upper temperature threshold, for any enabled monitored site.



**(6) uint8\_t tmu\_threshold\_config\_t::averageCriticalThresoldValue**

Valid when corresponding threshold is enabled. High temperature average critical threshold value. Determines the average upper critical temperature threshold, for any enabled monitored site.

**22.3.2 struct tmu\_interrupt\_status\_t****Data Fields**

- uint32\_t [interruptDetectMask](#)  
*The mask of interrupt status flags.*

**Field Documentation****(1) uint32\_t tmu\_interrupt\_status\_t::interruptDetectMask**

Refer to "\_tmu\_interrupt\_status\_flags" enumeration.

**22.3.3 struct tmu\_config\_t****Data Fields**

- [tmu\\_average\\_low\\_pass\\_filter\\_t](#) averageLPF  
*The average temperature is calculated as:  $ALPF \times Current\_Temp + (1 - ALPF) \times Average\_Temp$ .*

**Field Documentation****(1) tmu\_average\_low\_pass\_filter\_t tmu\_config\_t::averageLPF**

For proper operation, this field should only change when monitoring is disabled.

**22.4 Macro Definition Documentation****22.4.1 #define FSL\_TMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))**

Version 2.1.1.

**22.5 Enumeration Type Documentation****22.5.1 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. >

## 22.5.2 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)

## 22.5.3 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.

## 22.5.4 enum tmu\_amplifier\_gain\_t

Enumerator

***kTMU\_AmplifierGain6\_34*** TMU amplifier gain voltage 6.34mV.

***kTMU\_AmplifierGain6\_485*** TMU amplifier gain voltage 6.485mV.

***kTMU\_AmplifierGain6\_63*** TMU amplifier gain voltage 6.63mV.

***kTMU\_AmplifierGain6\_775*** TMU amplifier gain voltage 6.775mV.

***kTMU\_AmplifierGain6\_92*** TMU amplifier gain voltage 6.92mV.

***kTMU\_AmplifierGain7\_065*** TMU amplifier gain voltage 7.065mV.

***kTMU\_AmplifierGain7\_21*** TMU amplifier gain voltage 7.21mV.

***kTMU\_AmplifierGain7\_355*** TMU amplifier gain voltage 7.355mV.

***kTMU\_AmplifierGain7\_5*** TMU amplifier gain voltage 7.5mV.

***kTMU\_AmplifierGain7\_645*** TMU amplifier gain voltage 7.645mV.

***kTMU\_AmplifierGain7\_79*** TMU amplifier gain voltage 7.79mV.

***kTMU\_AmplifierGain7\_935*** TMU amplifier gain voltage 7.935mV.

***kTMU\_AmplifierGain8\_08*** TMU amplifier gain voltage 8.08mV(default).

***kTMU\_AmplifierGain8\_225*** TMU amplifier gain voltage 8.225mV.

*kTMU\_AmplifierGain8\_37* TMU amplifier gain voltage 8.37mV.

*kTMU\_AmplifierGain8\_515* TMU amplifier gain voltage 8.515mV.

## 22.5.5 enum tmu\_amplifier\_reference\_voltage\_t

Enumerator

*kTMU\_AmplifierReferenceVoltage510* TMU amplifier reference voltage 510mV.

*kTMU\_AmplifierReferenceVoltage517\_5* TMU amplifier reference voltage 517.5mV.

*kTMU\_AmplifierReferenceVoltage525* TMU amplifier reference voltage 525mV.

*kTMU\_AmplifierReferenceVoltage532\_5* TMU amplifier reference voltage 532.5mV.

*kTMU\_AmplifierReferenceVoltage540* TMU amplifier reference voltage 540mV.

*kTMU\_AmplifierReferenceVoltage547\_5* TMU amplifier reference voltage 547.5mV.

*kTMU\_AmplifierReferenceVoltage555* TMU amplifier reference voltage 555mV.

*kTMU\_AmplifierReferenceVoltage562\_5* TMU amplifier reference voltage 562.5mV.

*kTMU\_AmplifierReferenceVoltage570* TMU amplifier reference voltage 570mV.

*kTMU\_AmplifierReferenceVoltage577\_5* TMU amplifier reference voltage 577.5mV.

*kTMU\_AmplifierReferenceVoltage585* TMU amplifier reference voltage 585mV.

*kTMU\_AmplifierReferenceVoltage592\_5* TMU amplifier reference voltage 592.5mV.

*kTMU\_AmplifierReferenceVoltage600* TMU amplifier reference voltage 600mV.

*kTMU\_AmplifierReferenceVoltage607\_5* TMU amplifier reference voltage 607.5mV.

*kTMU\_AmplifierReferenceVoltage615* TMU amplifier reference voltage 615mV.

*kTMU\_AmplifierReferenceVoltage622\_5* TMU amplifier reference voltage 622.5mV.

*kTMU\_AmplifierReferenceVoltage630* TMU amplifier reference voltage 630mV.

*kTMU\_AmplifierReferenceVoltage637\_5* TMU amplifier reference voltage 637.5mV.

*kTMU\_AmplifierReferenceVoltage645* TMU amplifier reference voltage 645mV.

*kTMU\_AmplifierReferenceVoltage652\_5* TMU amplifier reference voltage 652.5mV(default).

*kTMU\_AmplifierReferenceVoltage660* TMU amplifier reference voltage 660mV.

*kTMU\_AmplifierReferenceVoltage667\_5* TMU amplifier reference voltage 667.5mV.

*kTMU\_AmplifierReferenceVoltage675* TMU amplifier reference voltage 675mV.

*kTMU\_AmplifierReferenceVoltage682\_5* TMU amplifier reference voltage 682.5mV.

*kTMU\_AmplifierReferenceVoltage690* TMU amplifier reference voltage 690mV.

*kTMU\_AmplifierReferenceVoltage697\_5* TMU amplifier reference voltage 697.5mV.

*kTMU\_AmplifierReferenceVoltage705* TMU amplifier reference voltage 705mV.

*kTMU\_AmplifierReferenceVoltage712\_5* TMU amplifier reference voltage 712.5mV.

*kTMU\_AmplifierReferenceVoltage720* TMU amplifier reference voltage 720mV.

*kTMU\_AmplifierReferenceVoltage727\_5* TMU amplifier reference voltage 727.5mV.

*kTMU\_AmplifierReferenceVoltage735* TMU amplifier reference voltage 735mV.

*kTMU\_AmplifierReferenceVoltage742\_5* TMU amplifier reference voltage 742.5mV.

## 22.6 Function Documentation

### 22.6.1 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. |

**22.6.2 void TMU\_Deinit ( TMU\_Type \* *base* )**

## Parameters

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

**22.6.3 void TMU\_GetDefaultConfig ( tmu\_config\_t \* *config* )**

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

Example:

```
config->averageLPF = kTMU_AverageLowPassFilter0_5;
```

## Parameters

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

**22.6.4 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. |

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

## Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | TMU peripheral base address.                                      |
| <i>mask</i> | The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration. |

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

## Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | TMU peripheral base address.                                      |
| <i>mask</i> | The interrupt mask. Refer to "_tmu_interrupt_enable" enumeration. |

### 22.6.7 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. |

### 22.6.8 void TMU\_ClearInterruptStatusFlags ( TMU\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                                                         |
|-------------|-----------------------------------------------------------------------------------------|
| <i>base</i> | TMU peripheral base address.                                                            |
| <i>mask</i> | The mask of interrupt status flags. Refer to "_tmu_interrupt_status_flags" enumeration. |

### 22.6.9 status\_t TMU\_GetImmediateTemperature ( TMU\_Type \* *base*, uint32\_t \* *temperature* )

## Parameters

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>base</i>        | TMU peripheral base address.                         |
| <i>temperature</i> | Last immediate temperature reading at site when V=1. |

## 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. |

### 22.6.10 `status_t TMU_GetAverageTemperature ( TMU_Type * base, uint32_t * temperature )`

## Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | TMU peripheral base address.              |
| <i>temperature</i> | Last average temperature reading at site. |

## 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. |

### 22.6.11 `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. |

## Chapter 23

# WDOG: Watchdog Timer Driver

### 23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

### 23.2 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, 1))  
*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)



- *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.*

## 23.3 Data Structure Documentation

### 23.3.1 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*

### 23.3.2 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.*

## Field Documentation

(1) bool `wdog_config_t::enableTimeOutAssert`

## 23.4 Enumeration Type Documentation

### 23.4.1 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.

### 23.4.2 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

## 23.5 Function Documentation

### 23.5.1 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;
```

```

* 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](#)

### 23.5.2 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    |

### 23.5.3 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).

### 23.5.4 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 |
|-------------|------------------------------|

**23.5.5 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 |
|-------------|------------------------------|

**23.5.6 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 |
|-------------|------------------------------|

**23.5.7 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 |
|-------------|------------------------------|

### 23.5.8 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> |

### 23.5.9 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.

### 23.5.10 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);
*
```

## 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 |

### 23.5.11 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. |

### 23.5.12 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. |

### 23.5.13 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 |
|-------------|------------------------------|

**23.5.14 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 |
|-------------|------------------------------|



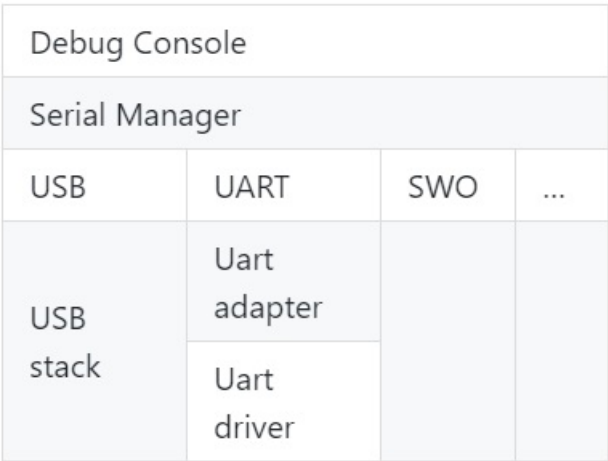
# Chapter 24

## Debug Console

### 24.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. The below picture shows the layout of debug console.



Debug console overview

### 24.2 Function groups

#### 24.2.1 Initialization

To initialize the debug console, call the [DbgConsole\\_Init\(\)](#) function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate,
 serial_port_type_t device, uint32_t clkSrcFreq);
```

Select the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
 kSerialPort_Uart = 1U,
 kSerialPort_UsbCdc,
 kSerialPort_Swo,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral.

This example shows how to call the `DbgConsole_Init()` given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,
 BOARD_DEBUG_UART_CLK_FREQ);
```

## 24.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype "`%[flags][width][.precision][length]specifier`", which is explained below

| flags   | Description                                                                                                                                                                                                                                                                                                                                                                             |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -       | Left-justified within the given field width. Right-justified is the default.                                                                                                                                                                                                                                                                                                            |
| +       | Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.                                                                                                                                                                                                                                |
| (space) | If no sign is written, a blank space is inserted before the value.                                                                                                                                                                                                                                                                                                                      |
| #       | Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed. |
| 0       | Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).                                                                                                                                                                                                                                                                           |

| Width    | Description                                                                                                                                                                                            |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (number) | A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger. |
| *        | The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                          |

| <b>.precision</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .number           | For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed. |
| .*                | The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| <b>length</b>  | <b>Description</b> |
|----------------|--------------------|
| Do not support |                    |

| <b>specifier</b> | <b>Description</b>                           |
|------------------|----------------------------------------------|
| d or i           | Signed decimal integer                       |
| f                | Decimal floating point                       |
| F                | Decimal floating point capital letters       |
| x                | Unsigned hexadecimal integer                 |
| X                | Unsigned hexadecimal integer capital letters |
| o                | Signed octal                                 |
| b                | Binary value                                 |
| p                | Pointer address                              |
| u                | Unsigned decimal integer                     |
| c                | Character                                    |
| s                | String of characters                         |
| n                | Nothing printed                              |

| specifier | Description |
|-----------|-------------|
|-----------|-------------|

- Support a format specifier for SCANF following this prototype " %[\*][width][length]specifier", which is explained below

| *                                                                                                                                                                | Description |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument. |             |

| width                                                                                        | Description |
|----------------------------------------------------------------------------------------------|-------------|
| This specifies the maximum number of characters to be read in the current reading operation. |             |

| length      | Description                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hh          | The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).                                                                     |
| h           | The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).                                                                    |
| l           | The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.           |
| ll          | The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s. |
| L           | The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).                                                                                            |
| j or z or t | Not supported                                                                                                                                                                                           |

| specifier              | Qualifying Input                                                                                                                                                                                                                                 | Type of argument |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| c                      | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char *           |
| i                      | Integer: : Number optionally preceded with a + or - sign                                                                                                                                                                                         | int *            |
| d                      | Decimal integer: Number optionally preceded with a + or - sign                                                                                                                                                                                   | int *            |
| a, A, e, E, f, F, g, G | Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4                      | float *          |
| o                      | Octal Integer:                                                                                                                                                                                                                                   | int *            |
| s                      | String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).                                                                                       | char *           |
| u                      | Unsigned decimal integer.                                                                                                                                                                                                                        | unsigned int *   |

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```

 version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
 toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */

```

### 24.2.3 SDK\_DEBUGCONSOLE and SDK\_DEBUGCONSOLE\_UART

There are two macros `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` added to configure `PRINTF` and low level output peripheral.

- The macro `SDK_DEBUGCONSOLE` is used for frontend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The function can be set by the macro `SDK_DEBUGCONSOLE`.
- The macro `SDK_DEBUGCONSOLE_UART` is used for backend. It is used to decide whether provide low level IO implementation to toolchain `printf` and `scanf`. For example, within MCU-Xpresso, if the macro `SDK_DEBUGCONSOLE_UART` is defined, `__sys_write` and `__sys_readc` will be used when `__REDLIB__` is defined; `_write` and `_read` will be used in other cases. The macro does not specifically refer to the peripheral "UART". It refers to the external peripheral similar to UART, like as USB CDC, UART, SWO, etc. So if the macro `SDK_DEBUGCONSOLE_UART` is not defined when tool-chain `printf` is calling, the semihosting will be used.

The following matrix shows the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

| <b>SDK_DEBUGCONSOLE</b>                      | <b>SDK_DEBUGCONSOLE_UART</b> | <b>PRINTF</b>         | <b>printf</b>        |
|----------------------------------------------|------------------------------|-----------------------|----------------------|
| DEBUGCONSOLE_-<br>REDIRECT_TO_SDK            | defined                      | Low level peripheral* | Low level peripheral |
| DEBUGCONSOLE_-<br>REDIRECT_TO_SDK            | undefined                    | Low level peripheral* | semihost             |
| DEBUGCONSOLE_-<br>REDIRECT_TO_TO-<br>OLCHAIN | defined                      | Low level peripheral* | Low level peripheral |
| DEBUGCONSOLE_-<br>REDIRECT_TO_TO-<br>OLCHAIN | undefined                    | semihost              | semihost             |
| DEBUGCONSOLE_-<br>DISABLE                    | defined                      | No output             | Low level peripheral |
| DEBUGCONSOLE_-<br>DISABLE                    | undefined                    | No output             | semihost             |

| SDK_DEBUGCONSOLE | SDK_DEBUGCONSOLE_UART | PRINTF | printf |
|------------------|-----------------------|--------|--------|
|------------------|-----------------------|--------|--------|

\* the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

## 24.3 Typical use case

### Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalent 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\r\nTime: %u ticks %2.5f milliseconds\r\n\r\nDONE\r\n\r\n", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

### Print out failure messages using MCUXpresso SDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
 PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
, line, func);
 for (;;)
 {}
}
```

**Note:**

To use 'printf' and 'scanf' for GNUC Base, add file 'fsl\_sbrk.c' in path: ..\{package}\devices\{subset}\utilities\fsl-\_sbrk.c to your project.

**Modules**

- [SWO](#)
- [Semihosting](#)
- [debug console configuration](#)

*The configuration is used for debug console only.*

**Macros**

- #define [DEBUGCONSOLE\\_REDIRECT\\_TO\\_TOOLCHAIN](#) 0U  
*Definition select redirect toolchain printf, scanf to uart or not.*
- #define [DEBUGCONSOLE\\_REDIRECT\\_TO\\_SDK](#) 1U  
*Select SDK version printf, scanf.*
- #define [DEBUGCONSOLE\\_DISABLE](#) 2U  
*Disable debugconsole function.*
- #define [SDK\\_DEBUGCONSOLE](#) [DEBUGCONSOLE\\_REDIRECT\\_TO\\_SDK](#)  
*Definition to select sdk or toolchain printf, scanf.*
- #define [PRINTF](#) [DbgConsole\\_Printf](#)  
*Definition to select redirect toolchain printf, scanf to uart or not.*

**Variables**

- [serial\\_handle\\_t](#) g\_serialHandle  
*serial manager handle*

**Initialization**

- [status\\_t](#) [DbgConsole\\_Init](#) (uint8\_t instance, uint32\_t baudRate, [serial\\_port\\_type\\_t](#) device, uint32\_t clkSrcFreq)  
*Initializes the peripheral used for debug messages.*
- [status\\_t](#) [DbgConsole\\_Deinit](#) (void)  
*De-initializes the peripheral used for debug messages.*
- [status\\_t](#) [DbgConsole\\_EnterLowpower](#) (void)  
*Prepares to enter low power consumption.*
- [status\\_t](#) [DbgConsole\\_ExitLowpower](#) (void)  
*Restores from low power consumption.*
- int [DbgConsole\\_Printf](#) (const char \*fmt\_s,...)  
*Writes formatted output to the standard output stream.*
- int [DbgConsole\\_Vprintf](#) (const char \*fmt\_s, va\_list formatStringArg)  
*Writes formatted output to the standard output stream.*
- int [DbgConsole\\_Putchar](#) (int ch)  
*Writes a character to stdout.*
- int [DbgConsole\\_Scanf](#) (char \*fmt\_s,...)  
*Reads formatted data from the standard input stream.*
- int [DbgConsole\\_Getchar](#) (void)



- *Reads a character from standard input.*  
int [DbgConsole\\_BlockingPrintf](#) (const char \*fmt\_s,...)  
*Writes formatted output to the standard output stream with the blocking mode.*
- int [DbgConsole\\_BlockingVprintf](#) (const char \*fmt\_s, va\_list formatStringArg)  
*Writes formatted output to the standard output stream with the blocking mode.*
- [status\\_t DbgConsole\\_Flush](#) (void)  
*Debug console flush.*
- [status\\_t DbgConsole\\_TryGetchar](#) (char \*ch)  
*Debug console try to get char This function provides a API which will not block current task, if character is available return it, otherwise return fail.*

## 24.4 Macro Definition Documentation

### 24.4.1 #define DEBUGCONSOLE\_REDIRECT\_TO\_TOOLCHAIN 0U

Select toolchain printf and scanf.

### 24.4.2 #define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U

### 24.4.3 #define DEBUGCONSOLE\_DISABLE 2U

### 24.4.4 #define SDK\_DEBUGCONSOLE DEBUGCONSOLE\_REDIRECT\_TO\_SDK

The macro only support to be redefined in project setting.

### 24.4.5 #define PRINTF DbgConsole\_Printf

if SDK\_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK\_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK\_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

## 24.5 Function Documentation

### 24.5.1 status\_t DbgConsole\_Init ( uint8\_t instance, uint32\_t baudRate, serial\_port\_type\_t device, uint32\_t clkSrcFreq )

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

## Parameters

|                   |                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>   | The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART peripheral is LPUART1. If the uart_adapter.c is added to the current project, the UART peripheral is UART1. |
| <i>baudRate</i>   | The desired baud rate in bits per second.                                                                                                                                                                                                                                                                                                                                                 |
| <i>device</i>     | Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"> <li>• kSerialPort_Uart,</li> <li>• kSerialPort_UsbCdc</li> </ul>                                                                                                                                                                                                             |
| <i>clkSrcFreq</i> | Frequency of peripheral source clock.                                                                                                                                                                                                                                                                                                                                                     |

## Returns

Indicates whether initialization was successful or not.

## Return values

|                        |                        |
|------------------------|------------------------|
| <i>kStatus_Success</i> | Execution successfully |
|------------------------|------------------------|

### 24.5.2 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.

### 24.5.3 status\_t DbgConsole\_EnterLowpower ( void )

This function is used to prepare to enter low power consumption.

## Returns

Indicates whether de-initialization was successful or not.

**24.5.4 status\_t DbgConsole\_ExitLowpower ( void )**

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

**24.5.5 int DbgConsole\_Printf ( const char \* *fmt\_s*, ... )**

Call this function to write a formatted output to the standard output stream.

Parameters

|              |                        |
|--------------|------------------------|
| <i>fmt_s</i> | Format control string. |
|--------------|------------------------|

Returns

Returns the number of characters printed or a negative value if an error occurs.

**24.5.6 int DbgConsole\_Vprintf ( const char \* *fmt\_s*, va\_list *formatStringArg* )**

Call this function to write a formatted output to the standard output stream.

Parameters

|                         |                        |
|-------------------------|------------------------|
| <i>fmt_s</i>            | Format control string. |
| <i>formatString-Arg</i> | Format arguments.      |

Returns

Returns the number of characters printed or a negative value if an error occurs.

**24.5.7 int DbgConsole\_Putchar ( int *ch* )**

Call this function to write a character to stdout.

## Parameters

|           |                          |
|-----------|--------------------------|
| <i>ch</i> | Character to be written. |
|-----------|--------------------------|

## Returns

Returns the character written.

### 24.5.8 int DbgConsole\_Scanf ( char \* *fmt\_s*, ... )

Call this function to read formatted data from the standard input stream.

## Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_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>fmt_s</i> | Format control string. |
|--------------|------------------------|

## Returns

Returns the number of fields successfully converted and assigned.

### 24.5.9 int DbgConsole\_Getchar ( void )

Call this function to read a character from standard input.

## Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_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.

### 24.5.10 int DbgConsole\_BlockingPrintf ( const char \* *fmt\_s*, ... )

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set or not. The function could be used in system ISR mode with DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set.

Parameters

|              |                        |
|--------------|------------------------|
| <i>fmt_s</i> | Format control string. |
|--------------|------------------------|

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 24.5.11 int DbgConsole\_BlockingVprintf ( const char \* *fmt\_s*, va\_list *formatStringArg* )

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set or not. The function could be used in system ISR mode with DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set.

Parameters

|                        |                        |
|------------------------|------------------------|
| <i>fmt_s</i>           | Format control string. |
| <i>formatStringArg</i> | Format arguments.      |

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 24.5.12 status\_t DbgConsole\_Flush ( void )

Call this function to wait the tx buffer empty. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

Returns

Indicates whether wait idle was successful or not.

### 24.5.13 status\_t DbgConsole\_TryGetchar ( char \* *ch* )

## Parameters

|           |                                |
|-----------|--------------------------------|
| <i>ch</i> | the address of char to receive |
|-----------|--------------------------------|

## Returns

Indicates get char was successful or not.

## 24.6 debug console configuration

The configuration is used for debug console only.

### 24.6.1 Overview

Please note, it is not used for debug console lite.

### Macros

- #define **DEBUG\_CONSOLE\_TRANSMIT\_BUFFER\_LEN** (512U)  
*If Non-blocking mode is needed, please define it at project setting, otherwise blocking mode is the default transfer mode.*
- #define **DEBUG\_CONSOLE\_RECEIVE\_BUFFER\_LEN** (1024U)  
*define the receive buffer length which is used to store the user input, buffer is enabled automatically when non-blocking transfer is using, This value will affect the RAM's utilization, should be set per platform's capability and software requirement.*
- #define **DEBUG\_CONSOLE\_TX\_RELIABLE\_ENABLE** (1U)  
*Whether enable the reliable TX function If the macro is zero, the reliable TX function of the debug console is disabled.*
- #define **DEBUG\_CONSOLE\_RX\_ENABLE** (1U)  
*Whether enable the RX function If the macro is zero, the receive function of the debug console is disabled.*
- #define **DEBUG\_CONSOLE\_PRINTF\_MAX\_LOG\_LEN** (128U)  
*define the MAX log length debug console support , that is when you call printf("log", x);, the log length can not bigger than this value.*
- #define **DEBUG\_CONSOLE\_SCANF\_MAX\_LOG\_LEN** (20U)  
*define the buffer support buffer scanf log length, that is when you call scanf("log", &x);, the log length can not bigger than this value.*
- #define **DEBUG\_CONSOLE\_SYNCHRONIZATION\_BM** 0  
*Debug console synchronization User should not change these macro for synchronization mode, but add the corresponding synchronization mechanism per different software environment.*
- #define **DEBUG\_CONSOLE\_SYNCHRONIZATION\_FREERTOS** 1  
*synchronization for freertos software*
- #define **DEBUG\_CONSOLE\_SYNCHRONIZATION\_MODE** **DEBUG\_CONSOLE\_SYNCHRONIZATION\_BM**  
*RTOS synchronization mechanism disable If not defined, default is enable, to avoid multitask log print mess.*
- #define **DEBUG\_CONSOLE\_ENABLE\_ECHO\_FUNCTION** 0  
*echo function support If you want to use the echo function, please define **DEBUG\_CONSOLE\_ENABLE\_ECHO** at your project setting.*
- #define **BOARD\_USE\_VIRTUALCOM** 0U  
*Definition to select virtual com(USB CDC) as the debug console.*



## 24.6.2 Macro Definition Documentation

### 24.6.2.1 #define DEBUG\_CONSOLE\_TRANSMIT\_BUFFER\_LEN (512U)

Warning: If you want to use non-blocking transfer, please make sure the corresponding IO interrupt is enable, otherwise there is no output. And non-blocking is combine with buffer, no matter bare-metal or rtos. Below shows how to configure in your project if you want to use non-blocking mode. For IAR, right click project and select "Options", define it in "C/C++ Compiler->Preprocessor->Defined symbols". For KEIL, click "Options for Target. . .", define it in "C/C++->Preprocessor Symbols->Define". For ARM-GCC, open CmakeLists.txt and add the following lines, "SET(CMAKE\_C\_FLAGS\_DEBUG "\${CMAKE\_C\_FLAGS\_DEBUG} -DDEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING)" for debug target. "SET(CMAKE\_C\_FLAGS\_RELEASE "\${CMAKE\_C\_FLAGS\_RELEASE} -DDEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING)" for release target. For MCUXpresso, right click project and select "Properties", define it in "C/C++ Build->Settings->MCU C Compiler->Preprocessor".

define the transmit buffer length which is used to store the multi task log, buffer is enabled automatically when non-blocking transfer is using, This value will affect the RAM's utilization, should be set per platform's capability and software requirement. If it is configured too small, log maybe missed, because the log will not be buffered if the buffer is full, and the print will return immediately with -1. And this value should be multiple of 4 to meet memory alignment.

### 24.6.2.2 #define DEBUG\_CONSOLE\_RECEIVE\_BUFFER\_LEN (1024U)

If it is configured too small, log maybe missed, because buffer will be overwritten if buffer is too small. And this value should be multiple of 4 to meet memory alignment.

### 24.6.2.3 #define DEBUG\_CONSOLE\_TX\_RELIABLE\_ENABLE (1U)

When the macro is zero, the string of PRINTF will be thrown away after the transmit buffer is full.

### 24.6.2.4 #define DEBUG\_CONSOLE\_PRINTF\_MAX\_LOG\_LEN (128U)

This macro decide the local log buffer length, the buffer locate at stack, the stack maybe overflow if the buffer is too big and current task stack size not big enough.

### 24.6.2.5 #define DEBUG\_CONSOLE\_SCANF\_MAX\_LOG\_LEN (20U)

As same as the DEBUG\_CONSOLE\_BUFFER\_PRINTF\_MAX\_LOG\_LEN.

#### 24.6.2.6 **#define DEBUG\_CONSOLE\_SYNCHRONIZATION\_BM 0**

Such as, if another RTOS is used, add: `#define DEBUG_CONSOLE_SYNCHRONIZATION_XXXX 3` in this configuration file and implement the synchronization in `fsl.log.c`.

synchronization for baremetal software

#### 24.6.2.7 **#define DEBUG\_CONSOLE\_SYNCHRONIZATION\_MODE DEBUG\_CONSOLE\_SYNCHRONIZATION\_BM**

If other RTOS is used, you can implement the RTOS's specific synchronization mechanism in `fsl.log.c`. If synchronization is disabled, log maybe messed on terminal.

#### 24.6.2.8 **#define BOARD\_USE\_VIRTUALCOM 0U**

## 24.7 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.

### 24.7.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the `SDK_DEBUGCONSOLE` is `DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN`.

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

#### Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

#### Step 3: Starting semihosting

1. Choose "Semihosting\_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting. Please Make sure the `SDK_DEBUGCONSOLE_UART` is not defined in project settings.
4. Start the project by choosing Project>Download and Debug.
5. Choose View>Terminal I/O to display the output from the I/O operations.

### 24.7.2 Guide Semihosting for Keil µVision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

### 24.7.3 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.

### 24.7.4 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__heap_size__=0x2000")
```

## Step 2: Building the project

1. Change "CMakeLists.txt":

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"

**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

**Replace paragraph**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-common")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffunction-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fdata-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffreestanding")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-builtin")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mthumb")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mapcs")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --gc-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -static")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -z")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} muldefs")

**To**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --specs=rdimon.specs ")

**Remove**

target\_link\_libraries(semihosting\_ARMGCC.elf debug nosys)

2. Run "build\_debug.bat" to build project

### Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

2. After the setting, press "enter". The PuTTY window now shows the printf() output.

## 24.8 SWO

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDEs also support SWO, such IAR and KEIL, both input and output are supported, see below for details.

### 24.8.1 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 SERIAL\_PORT\_TYPE\_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(instance, baudRate, kSerialPort_Swo, clkSrcFreq);
```

3. Use PRINTF or printf to print some thing in application.

#### Step 2: Building the project

#### Step 3: Download and run project

##### 24.8.1.1 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.

## **24.8.2 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 O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

### **Step 3: Building the project**

1. Compile and link the project by choosing Project>Build Target or using F7.

### **Step 4: Run the project**

1. Choose "Debug" on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.



### **24.8.3 Guide SWO for MCUXpresso IDE**

**NOTE:** MCUX support SWO for LPC-Link2 debug probe only.

### **24.8.4 Guide SWO for ARMGCC**

**NOTE:** ARMGCC has no library support SWO.

## Chapter 25

# CODEC Driver

### 25.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

#### Modules

- [AK4497 Driver](#)
- [CODEC Common Driver](#)
- [CODEC I2C Driver](#)
- [WM8524 Driver](#)

## 25.2 CODEC Common Driver

### 25.2.1 Overview

The codec common driver provides a codec control abstraction interface.

#### Modules

- [AK4497 Adapter](#)
- [CODEC Adapter](#)
- [WM8524 Adapter](#)

#### Data Structures

- struct [codec\\_config\\_t](#)  
*Initialize structure of the codec. [More...](#)*
- struct [codec\\_capability\\_t](#)  
*codec capability [More...](#)*
- struct [codec\\_handle\\_t](#)  
*Codec handle definition. [More...](#)*

#### Macros

- `#define CODEC\_VOLUME\_MAX\_VALUE (100U)`  
*codec maximum volume range*

#### Enumerations

- enum {  
    [kStatus\\_CODEC\\_NotSupport](#) = MAKE\_STATUS(kStatusGroup\_CODEC, 0U),  
    [kStatus\\_CODEC\\_DeviceNotRegistered](#) = MAKE\_STATUS(kStatusGroup\_CODEC, 1U),  
    [kStatus\\_CODEC\\_I2CBusInitialFailed](#),  
    [kStatus\\_CODEC\\_I2CCommandTransferFailed](#) }  
    *CODEC status.*
- enum [codec\\_audio\\_protocol\\_t](#) {  
    [kCODEC\\_BusI2S](#) = 0U,  
    [kCODEC\\_BusLeftJustified](#) = 1U,  
    [kCODEC\\_BusRightJustified](#) = 2U,  
    [kCODEC\\_BusPCMA](#) = 3U,  
    [kCODEC\\_BusPCMB](#) = 4U,  
    [kCODEC\\_BusTDM](#) = 5U }  
    *AUDIO format definition.*

- enum {
  - kCODEC\_AudioSampleRate8KHz = 8000U,
  - kCODEC\_AudioSampleRate11025Hz = 11025U,
  - kCODEC\_AudioSampleRate12KHz = 12000U,
  - kCODEC\_AudioSampleRate16KHz = 16000U,
  - kCODEC\_AudioSampleRate22050Hz = 22050U,
  - kCODEC\_AudioSampleRate24KHz = 24000U,
  - kCODEC\_AudioSampleRate32KHz = 32000U,
  - kCODEC\_AudioSampleRate44100Hz = 44100U,
  - kCODEC\_AudioSampleRate48KHz = 48000U,
  - kCODEC\_AudioSampleRate96KHz = 96000U,
  - kCODEC\_AudioSampleRate192KHz = 192000U,
  - kCODEC\_AudioSampleRate384KHz = 384000U }*audio sample rate definition*
- enum {
  - kCODEC\_AudioBitWidth16bit = 16U,
  - kCODEC\_AudioBitWidth20bit = 20U,
  - kCODEC\_AudioBitWidth24bit = 24U,
  - kCODEC\_AudioBitWidth32bit = 32U }*audio bit width*
- enum codec\_module\_t {
  - kCODEC\_ModuleADC = 0U,
  - kCODEC\_ModuleDAC = 1U,
  - kCODEC\_ModulePGA = 2U,
  - kCODEC\_ModuleHeadphone = 3U,
  - kCODEC\_ModuleSpeaker = 4U,
  - kCODEC\_ModuleLinein = 5U,
  - kCODEC\_ModuleLineout = 6U,
  - kCODEC\_ModuleVref = 7U,
  - kCODEC\_ModuleMicbias = 8U,
  - kCODEC\_ModuleMic = 9U,
  - kCODEC\_ModuleI2SIn = 10U,
  - kCODEC\_ModuleI2SOut = 11U,
  - kCODEC\_ModuleMixer = 12U }*audio codec module*
- enum codec\_module\_ctrl\_cmd\_t { kCODEC\_ModuleSwitchI2SInInterface = 0U }
- audio codec module control cmd*
- enum {
  - kCODEC\_ModuleI2SInInterfacePCM = 0U,
  - kCODEC\_ModuleI2SInInterfaceDSD = 1U }*audio codec module digital interface*
- enum {
  - kCODEC\_RecordSourceDifferentialLine = 1U,
  - kCODEC\_RecordSourceLineInput = 2U,
  - kCODEC\_RecordSourceDifferentialMic = 4U,
  - kCODEC\_RecordSourceDigitalMic = 8U,

```
kCODEC_RecordSourceSingleEndMic = 16U }
```

*audio codec module record source value*

- enum {
  - kCODEC\_RecordChannelLeft1 = 1U,
  - kCODEC\_RecordChannelLeft2 = 2U,
  - kCODEC\_RecordChannelLeft3 = 4U,
  - kCODEC\_RecordChannelRight1 = 1U,
  - kCODEC\_RecordChannelRight2 = 2U,
  - kCODEC\_RecordChannelRight3 = 4U,
  - kCODEC\_RecordChannelDifferentialPositive1 = 1U,
  - kCODEC\_RecordChannelDifferentialPositive2 = 2U,
  - kCODEC\_RecordChannelDifferentialPositive3 = 4U,
  - kCODEC\_RecordChannelDifferentialNegative1 = 8U,
  - kCODEC\_RecordChannelDifferentialNegative2 = 16U,
  - kCODEC\_RecordChannelDifferentialNegative3 = 32U }

*audio codec record channel*

- enum {
  - kCODEC\_PlaySourcePGA = 1U,
  - kCODEC\_PlaySourceInput = 2U,
  - kCODEC\_PlaySourceDAC = 4U,
  - kCODEC\_PlaySourceMixerIn = 1U,
  - kCODEC\_PlaySourceMixerInLeft = 2U,
  - kCODEC\_PlaySourceMixerInRight = 4U,
  - kCODEC\_PlaySourceAux = 8U }

*audio codec module play source value*

- enum {
  - kCODEC\_PlayChannelHeadphoneLeft = 1U,
  - kCODEC\_PlayChannelHeadphoneRight = 2U,
  - kCODEC\_PlayChannelSpeakerLeft = 4U,
  - kCODEC\_PlayChannelSpeakerRight = 8U,
  - kCODEC\_PlayChannelLineOutLeft = 16U,
  - kCODEC\_PlayChannelLineOutRight = 32U,
  - kCODEC\_PlayChannelLeft0 = 1U,
  - kCODEC\_PlayChannelRight0 = 2U,
  - kCODEC\_PlayChannelLeft1 = 4U,
  - kCODEC\_PlayChannelRight1 = 8U,
  - kCODEC\_PlayChannelLeft2 = 16U,
  - kCODEC\_PlayChannelRight2 = 32U,
  - kCODEC\_PlayChannelLeft3 = 64U,
  - kCODEC\_PlayChannelRight3 = 128U }

*codec play channel*

- enum {

```

kCODEC_VolumeHeadphoneLeft = 1U,
kCODEC_VolumeHeadphoneRight = 2U,
kCODEC_VolumeSpeakerLeft = 4U,
kCODEC_VolumeSpeakerRight = 8U,
kCODEC_VolumeLineOutLeft = 16U,
kCODEC_VolumeLineOutRight = 32U,
kCODEC_VolumeLeft0 = 1UL << 0U,
kCODEC_VolumeRight0 = 1UL << 1U,
kCODEC_VolumeLeft1 = 1UL << 2U,
kCODEC_VolumeRight1 = 1UL << 3U,
kCODEC_VolumeLeft2 = 1UL << 4U,
kCODEC_VolumeRight2 = 1UL << 5U,
kCODEC_VolumeLeft3 = 1UL << 6U,
kCODEC_VolumeRight3 = 1UL << 7U,
kCODEC_VolumeDAC = 1UL << 8U }

```

*codec volume setting*

- enum {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

*audio codec capability*

## Functions

- `status_t CODEC_Init` (`codec_handle_t *handle`, `codec_config_t *config`)  
*Codec initialization.*
- `status_t CODEC_Deinit` (`codec_handle_t *handle`)  
*Codec de-initialization.*
- `status_t CODEC_SetFormat` (`codec_handle_t *handle`, `uint32_t mclk`, `uint32_t sampleRate`, `uint32_t bitWidth`)  
*set audio data format.*
- `status_t CODEC_ModuleControl` (`codec_handle_t *handle`, `codec_module_ctrl_cmd_t cmd`, `uint32_t data`)  
*codec module control.*
- `status_t CODEC_SetVolume` (`codec_handle_t *handle`, `uint32_t channel`, `uint32_t volume`)  
*set audio codec pl volume.*
- `status_t CODEC_SetMute` (`codec_handle_t *handle`, `uint32_t channel`, `bool mute`)  
*set audio codec module mute.*
- `status_t CODEC_SetPower` (`codec_handle_t *handle`, `codec_module_t module`, `bool powerOn`)  
*set audio codec power.*
- `status_t CODEC_SetRecord` (`codec_handle_t *handle`, `uint32_t recordSource`)  
*codec set record source.*
- `status_t CODEC_SetRecordChannel` (`codec_handle_t *handle`, `uint32_t leftRecordChannel`, `uint32_t rightRecordChannel`)  
*codec set record channel.*
- `status_t CODEC_SetPlay` (`codec_handle_t *handle`, `uint32_t playSource`)  
*codec set play source.*

## Driver version

- `#define FSL_CODEC_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 1)`)  
*CLOCK driver version 2.3.1.*

## 25.2.2 Data Structure Documentation

### 25.2.2.1 struct codec\_config\_t

#### Data Fields

- `uint32_t codecDevType`  
*codec type*
- `void * codecDevConfig`  
*Codec device specific configuration.*



### 25.2.2.2 struct codec\_capability\_t

#### Data Fields

- uint32\_t [codecModuleCapability](#)  
*codec module capability*
- uint32\_t [codecPlayCapability](#)  
*codec play capability*
- uint32\_t [codecRecordCapability](#)  
*codec record capability*
- uint32\_t [codecVolumeCapability](#)  
*codec volume capability*

### 25.2.2.3 struct \_codec\_handle

codec handle declaration

- Application should allocate a buffer with CODEC\_HANDLE\_SIZE for handle definition, such as uint8\_t codecHandleBuffer[CODEC\_HANDLE\_SIZE]; codec\_handle\_t \*codecHandle = codecHandleBuffer;

#### Data Fields

- [codec\\_config\\_t](#) \* [codecConfig](#)  
*codec configuration function pointer*
- const [codec\\_capability\\_t](#) \* [codecCapability](#)  
*codec capability*
- uint8\_t [codecDevHandle](#) [HAL\_CODEC\_HANDLER\_SIZE]  
*codec device handle*

## 25.2.3 Macro Definition Documentation

### 25.2.3.1 #define FSL\_CODEC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

## 25.2.4 Enumeration Type Documentation

### 25.2.4.1 anonymous enum

Enumerator

***kStatus\_CODEC\_NotSupport*** CODEC not support status.

***kStatus\_CODEC\_DeviceNotRegistered*** CODEC device register failed status.

***kStatus\_CODEC\_I2CBusInitialFailed*** CODEC i2c bus initialization failed status.

***kStatus\_CODEC\_I2CCommandTransferFailed*** CODEC i2c bus command transfer failed status.

#### 25.2.4.2 enum codec\_audio\_protocol\_t

Enumerator

*kCODEC\_BusI2S* I2S type.  
*kCODEC\_BusLeftJustified* Left justified mode.  
*kCODEC\_BusRightJustified* Right justified mode.  
*kCODEC\_BusPCMA* DSP/PCM A mode.  
*kCODEC\_BusPCMB* DSP/PCM B mode.  
*kCODEC\_BusTDM* TDM mode.

#### 25.2.4.3 anonymous enum

Enumerator

*kCODEC\_AudioSampleRate8KHz* Sample rate 8000 Hz.  
*kCODEC\_AudioSampleRate11025Hz* Sample rate 11025 Hz.  
*kCODEC\_AudioSampleRate12KHz* Sample rate 12000 Hz.  
*kCODEC\_AudioSampleRate16KHz* Sample rate 16000 Hz.  
*kCODEC\_AudioSampleRate22050Hz* Sample rate 22050 Hz.  
*kCODEC\_AudioSampleRate24KHz* Sample rate 24000 Hz.  
*kCODEC\_AudioSampleRate32KHz* Sample rate 32000 Hz.  
*kCODEC\_AudioSampleRate44100Hz* Sample rate 44100 Hz.  
*kCODEC\_AudioSampleRate48KHz* Sample rate 48000 Hz.  
*kCODEC\_AudioSampleRate96KHz* Sample rate 96000 Hz.  
*kCODEC\_AudioSampleRate192KHz* Sample rate 192000 Hz.  
*kCODEC\_AudioSampleRate384KHz* Sample rate 384000 Hz.

#### 25.2.4.4 anonymous enum

Enumerator

*kCODEC\_AudioBitWidth16bit* audio bit width 16  
*kCODEC\_AudioBitWidth20bit* audio bit width 20  
*kCODEC\_AudioBitWidth24bit* audio bit width 24  
*kCODEC\_AudioBitWidth32bit* audio bit width 32

#### 25.2.4.5 enum codec\_module\_t

Enumerator

*kCODEC\_ModuleADC* codec module ADC  
*kCODEC\_ModuleDAC* codec module DAC  
*kCODEC\_ModulePGA* codec module PGA  
*kCODEC\_ModuleHeadphone* codec module headphone

***kCODEC\_ModuleSpeaker*** codec module speaker  
***kCODEC\_ModuleLinein*** codec module linein  
***kCODEC\_ModuleLineout*** codec module lineout  
***kCODEC\_ModuleVref*** codec module VREF  
***kCODEC\_ModuleMicbias*** codec module MIC BIAS  
***kCODEC\_ModuleMic*** codec module MIC  
***kCODEC\_ModuleI2SIn*** codec module I2S in  
***kCODEC\_ModuleI2SOut*** codec module I2S out  
***kCODEC\_ModuleMixer*** codec module mixer

#### 25.2.4.6 enum codec\_module\_ctrl\_cmd\_t

Enumerator

***kCODEC\_ModuleSwitchI2SInInterface*** module digital interface swtch.

#### 25.2.4.7 anonymous enum

Enumerator

***kCODEC\_ModuleI2SInInterfacePCM*** Pcm interface.  
***kCODEC\_ModuleI2SInInterfaceDSD*** DSD interface.

#### 25.2.4.8 anonymous enum

Enumerator

***kCODEC\_RecordSourceDifferentialLine*** record source from differential line  
***kCODEC\_RecordSourceLineInput*** record source from line input  
***kCODEC\_RecordSourceDifferentialMic*** record source from differential mic  
***kCODEC\_RecordSourceDigitalMic*** record source from digital microphone  
***kCODEC\_RecordSourceSingleEndMic*** record source from single microphone

#### 25.2.4.9 anonymous enum

Enumerator

***kCODEC\_RecordChannelLeft1*** left record channel 1  
***kCODEC\_RecordChannelLeft2*** left record channel 2  
***kCODEC\_RecordChannelLeft3*** left record channel 3  
***kCODEC\_RecordChannelRight1*** right record channel 1  
***kCODEC\_RecordChannelRight2*** right record channel 2  
***kCODEC\_RecordChannelRight3*** right record channel 3  
***kCODEC\_RecordChannelDifferentialPositive1*** differential positive record channel 1

|                                                  |                                        |
|--------------------------------------------------|----------------------------------------|
| <i>kCODEC_RecordChannelDifferentialPositive2</i> | differential positive record channel 2 |
| <i>kCODEC_RecordChannelDifferentialPositive3</i> | differential positive record channel 3 |
| <i>kCODEC_RecordChannelDifferentialNegative1</i> | differential negative record channel 1 |
| <i>kCODEC_RecordChannelDifferentialNegative2</i> | differential negative record channel 2 |
| <i>kCODEC_RecordChannelDifferentialNegative3</i> | differential negative record channel 3 |

#### 25.2.4.10 anonymous enum

Enumerator

|                                      |                             |
|--------------------------------------|-----------------------------|
| <i>kCODEC_PlaySourcePGA</i>          | play source PGA, bypass ADC |
| <i>kCODEC_PlaySourceInput</i>        | play source Input3          |
| <i>kCODEC_PlaySourceDAC</i>          | play source DAC             |
| <i>kCODEC_PlaySourceMixerIn</i>      | play source mixer in        |
| <i>kCODEC_PlaySourceMixerInLeft</i>  | play source mixer in left   |
| <i>kCODEC_PlaySourceMixerInRight</i> | play source mixer in right  |
| <i>kCODEC_PlaySourceAux</i>          | play source mixer in AUx    |

#### 25.2.4.11 anonymous enum

Enumerator

|                                         |                              |
|-----------------------------------------|------------------------------|
| <i>kCODEC_PlayChannelHeadphoneLeft</i>  | play channel headphone left  |
| <i>kCODEC_PlayChannelHeadphoneRight</i> | play channel headphone right |
| <i>kCODEC_PlayChannelSpeakerLeft</i>    | play channel speaker left    |
| <i>kCODEC_PlayChannelSpeakerRight</i>   | play channel speaker right   |
| <i>kCODEC_PlayChannelLineOutLeft</i>    | play channel lineout left    |
| <i>kCODEC_PlayChannelLineOutRight</i>   | play channel lineout right   |
| <i>kCODEC_PlayChannelLeft0</i>          | play channel left0           |
| <i>kCODEC_PlayChannelRight0</i>         | play channel right0          |
| <i>kCODEC_PlayChannelLeft1</i>          | play channel left1           |
| <i>kCODEC_PlayChannelRight1</i>         | play channel right1          |
| <i>kCODEC_PlayChannelLeft2</i>          | play channel left2           |
| <i>kCODEC_PlayChannelRight2</i>         | play channel right2          |
| <i>kCODEC_PlayChannelLeft3</i>          | play channel left3           |
| <i>kCODEC_PlayChannelRight3</i>         | play channel right3          |

#### 25.2.4.12 anonymous enum

Enumerator

|                                    |                        |
|------------------------------------|------------------------|
| <i>kCODEC_VolumeHeadphoneLeft</i>  | headphone left volume  |
| <i>kCODEC_VolumeHeadphoneRight</i> | headphone right volume |
| <i>kCODEC_VolumeSpeakerLeft</i>    | speaker left volume    |
| <i>kCODEC_VolumeSpeakerRight</i>   | speaker right volume   |

***kCODEC\_VolumeLineOutLeft*** lineout left volume  
***kCODEC\_VolumeLineOutRight*** lineout right volume  
***kCODEC\_VolumeLeft0*** left0 volume  
***kCODEC\_VolumeRight0*** right0 volume  
***kCODEC\_VolumeLeft1*** left1 volume  
***kCODEC\_VolumeRight1*** right1 volume  
***kCODEC\_VolumeLeft2*** left2 volume  
***kCODEC\_VolumeRight2*** right2 volume  
***kCODEC\_VolumeLeft3*** left3 volume  
***kCODEC\_VolumeRight3*** right3 volume  
***kCODEC\_VolumeDAC*** dac volume

#### 25.2.4.13 anonymous enum

Enumerator

***kCODEC\_SupportModuleADC*** codec capability of module ADC  
***kCODEC\_SupportModuleDAC*** codec capability of module DAC  
***kCODEC\_SupportModulePGA*** codec capability of module PGA  
***kCODEC\_SupportModuleHeadphone*** codec capability of module headphone  
***kCODEC\_SupportModuleSpeaker*** codec capability of module speaker  
***kCODEC\_SupportModuleLinein*** codec capability of module linein  
***kCODEC\_SupportModuleLineout*** codec capability of module lineout  
***kCODEC\_SupportModuleVref*** codec capability of module vref  
***kCODEC\_SupportModuleMicbias*** codec capability of module mic bias  
***kCODEC\_SupportModuleMic*** codec capability of module mic bias  
***kCODEC\_SupportModuleI2SIn*** codec capability of module I2S in  
***kCODEC\_SupportModuleI2SOut*** codec capability of module I2S out  
***kCODEC\_SupportModuleMixer*** codec capability of module mixer  
***kCODEC\_SupportModuleI2SInSwitchInterface*** codec capability of module I2S in switch interface  
  
***kCODEC\_SupportPlayChannelLeft0*** codec capability of play channel left 0  
***kCODEC\_SupportPlayChannelRight0*** codec capability of play channel right 0  
***kCODEC\_SupportPlayChannelLeft1*** codec capability of play channel left 1  
***kCODEC\_SupportPlayChannelRight1*** codec capability of play channel right 1  
***kCODEC\_SupportPlayChannelLeft2*** codec capability of play channel left 2  
***kCODEC\_SupportPlayChannelRight2*** codec capability of play channel right 2  
***kCODEC\_SupportPlayChannelLeft3*** codec capability of play channel left 3  
***kCODEC\_SupportPlayChannelRight3*** codec capability of play channel right 3  
***kCODEC\_SupportPlaySourcePGA*** codec capability of set playback source PGA  
***kCODEC\_SupportPlaySourceInput*** codec capability of set playback source INPUT  
***kCODEC\_SupportPlaySourceDAC*** codec capability of set playback source DAC  
***kCODEC\_SupportPlaySourceMixerIn*** codec capability of set play source Mixer in  
***kCODEC\_SupportPlaySourceMixerInLeft*** codec capability of set play source Mixer in left  
***kCODEC\_SupportPlaySourceMixerInRight*** codec capability of set play source Mixer in right

***kCODEC\_SupportPlaySourceAux*** codec capability of set play source aux

***kCODEC\_SupportRecordSourceDifferentialLine*** codec capability of record source differential line

***kCODEC\_SupportRecordSourceLineInput*** codec capability of record source line input

***kCODEC\_SupportRecordSourceDifferentialMic*** codec capability of record source differential mic

***kCODEC\_SupportRecordSourceDigitalMic*** codec capability of record digital mic

***kCODEC\_SupportRecordSourceSingleEndMic*** codec capability of single end mic

***kCODEC\_SupportRecordChannelLeft1*** left record channel 1

***kCODEC\_SupportRecordChannelLeft2*** left record channel 2

***kCODEC\_SupportRecordChannelLeft3*** left record channel 3

***kCODEC\_SupportRecordChannelRight1*** right record channel 1

***kCODEC\_SupportRecordChannelRight2*** right record channel 2

***kCODEC\_SupportRecordChannelRight3*** right record channel 3

## 25.2.5 Function Documentation

### 25.2.5.1 **status\_t CODEC\_Init ( codec\_handle\_t \* *handle*, codec\_config\_t \* *config* )**

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | codec handle.         |
| <i>config</i> | codec configurations. |

Returns

kStatus\_Success is success, else de-initial failed.

### 25.2.5.2 **status\_t CODEC\_Deinit ( codec\_handle\_t \* *handle* )**

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 25.2.5.3 **status\_t CODEC\_SetFormat ( codec\_handle\_t \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* )**

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.2.5.4 status\_t CODEC\_ModuleControl ( codec\_handle\_t \* *handle*, codec\_module\_ctrl\_cmd\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature.

## Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.2.5.5 status\_t CODEC\_SetVolume ( codec\_handle\_t \* *handle*, uint32\_t *channel*, uint32\_t *volume* )

## Parameters

|                |                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                                              |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of <code>_codec_volume_capability</code> or <code>_codec_play_channel</code> . |
| <i>volume</i>  | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.                                                                 |

Returns

`kStatus_Success` is success, else configure failed.

#### 25.2.5.6 `status_t CODEC_SetMute ( codec_handle_t * handle, uint32_t channel, bool mute )`

Parameters

|                |                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | codec handle.                                                                                                                              |
| <i>channel</i> | audio codec volume channel, can be a value or combine value of <code>_codec_volume_capability</code> or <code>_codec_play_channel</code> . |
| <i>mute</i>    | true is mute, false is unmute.                                                                                                             |

Returns

`kStatus_Success` is success, else configure failed.

#### 25.2.5.7 `status_t CODEC_SetPower ( codec_handle_t * handle, codec_module_t module, bool powerOn )`

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

`kStatus_Success` is success, else configure failed.

#### 25.2.5.8 `status_t CODEC_SetRecord ( codec_handle_t * handle, uint32_t recordSource )`



## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

### 25.2.5.9 `status_t CODEC_SetRecordChannel ( codec_handle_t * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel )`

## Parameters

|                            |                                                                                                                                                     |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                                       |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value combine of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value combine of member in <code>_codec_record_channel</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

### 25.2.5.10 `status_t CODEC_SetPlay ( codec_handle_t * handle, uint32_t playSource )`

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

## 25.3 CODEC I2C Driver

### 25.3.1 Overview

The codec common driver provides a codec control abstraction interface.

### Data Structures

- struct `codec_i2c_config_t`  
CODEC I2C configurations structure. [More...](#)

### Macros

- #define `CODEC_I2C_MASTER_HANDLER_SIZE` `HAL_I2C_MASTER_HANDLE_SIZE`  
codec i2c handler

### Enumerations

- enum `codec_reg_addr_t` {  
  `kCODEC_RegAddr8Bit` = 1U,  
  `kCODEC_RegAddr16Bit` = 2U }  
CODEC device register address type.
- enum `codec_reg_width_t` {  
  `kCODEC_RegWidth8Bit` = 1U,  
  `kCODEC_RegWidth16Bit` = 2U,  
  `kCODEC_RegWidth32Bit` = 4U }  
CODEC device register width.

### Functions

- `status_t CODEC_I2C_Init` (void \*handle, uint32\_t i2cInstance, uint32\_t i2cBaudrate, uint32\_t i2cSourceClockHz)  
CODEC i2c bus initialization.
- `status_t CODEC_I2C_Deinit` (void \*handle)  
CODEC i2c de-initialization.
- `status_t CODEC_I2C_Send` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*txBuff, uint8\_t txBuffSize)  
codec i2c send function.
- `status_t CODEC_I2C_Receive` (void \*handle, uint8\_t deviceAddress, uint32\_t subAddress, uint8\_t subaddressSize, uint8\_t \*rxBuff, uint8\_t rxBuffSize)  
codec i2c receive function.

## 25.3.2 Data Structure Documentation

### 25.3.2.1 struct codec\_i2c\_config\_t

#### Data Fields

- uint32\_t [codecI2CInstance](#)  
*i2c bus instance*
- uint32\_t [codecI2CSourceClock](#)  
*i2c bus source clock frequency*

## 25.3.3 Enumeration Type Documentation

### 25.3.3.1 enum codec\_reg\_addr\_t

#### Enumerator

- kCODEC\_RegAddr8Bit*** 8-bit register address.  
***kCODEC\_RegAddr16Bit*** 16-bit register address.

### 25.3.3.2 enum codec\_reg\_width\_t

#### Enumerator

- kCODEC\_RegWidth8Bit*** 8-bit register width.  
***kCODEC\_RegWidth16Bit*** 16-bit register width.  
***kCODEC\_RegWidth32Bit*** 32-bit register width.

## 25.3.4 Function Documentation

### 25.3.4.1 status\_t CODEC\_I2C\_Init ( void \* *handle*, uint32\_t *i2cInstance*, uint32\_t *i2cBaudrate*, uint32\_t *i2cSourceClockHz* )

#### Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>handle</i>      | i2c master handle.                                                  |
| <i>i2cInstance</i> | instance number of the i2c bus, such as 0 is corresponding to I2C0. |

|                          |                             |
|--------------------------|-----------------------------|
| <i>i2cBaudrate</i>       | i2c baudrate.               |
| <i>i2cSource-ClockHz</i> | i2c source clock frequency. |

Returns

kStatus\_HAL\_I2cSuccess is success, else initial failed.

#### 25.3.4.2 status\_t CODEC\_I2C\_Deinit ( void \* *handle* )

Parameters

|               |                    |
|---------------|--------------------|
| <i>handle</i> | i2c master handle. |
|---------------|--------------------|

Returns

kStatus\_HAL\_I2cSuccess is success, else deinitial failed.

#### 25.3.4.3 status\_t CODEC\_I2C\_Send ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *txBuff*, uint8\_t *txBuffSize* )

Parameters

|                       |                         |
|-----------------------|-------------------------|
| <i>handle</i>         | i2c master handle.      |
| <i>deviceAddress</i>  | codec device address.   |
| <i>subAddress</i>     | register address.       |
| <i>subaddressSize</i> | register address width. |
| <i>txBuff</i>         | tx buffer pointer.      |
| <i>txBuffSize</i>     | tx buffer size.         |

Returns

kStatus\_HAL\_I2cSuccess is success, else send failed.

#### 25.3.4.4 status\_t CODEC\_I2C\_Receive ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *rxBuff*, uint8\_t *rxBuffSize* )

## Parameters

|                       |                         |
|-----------------------|-------------------------|
| <i>handle</i>         | i2c master handle.      |
| <i>deviceAddress</i>  | codec device address.   |
| <i>subAddress</i>     | register address.       |
| <i>subaddressSize</i> | register address width. |
| <i>rxBuff</i>         | rx buffer pointer.      |
| <i>rxBuffSize</i>     | rx buffer size.         |

## Returns

kStatus\_HAL\_I2cSuccess is success, else receive failed.

## 25.4 AK4497 Driver

### 25.4.1 Overview

The ak4497 driver provides a codec control interface.

### Data Structures

- struct [ak4497\\_dsd\\_config\\_t](#)  
*Initialize DSD mode structure of AK4497. [More...](#)*
- struct [ak4497\\_pcm\\_config\\_t](#)  
*Initialize PCM mode structure of AK4497. [More...](#)*
- struct [ak4497\\_config\\_t](#)  
*Initialize structure of AK4497. [More...](#)*
- struct [ak4497\\_handle\\_t](#)  
*ak4497 codec handler [More...](#)*

### Macros

- #define [AK4497\\_I2C\\_HANDLER\\_SIZE](#) [CODEC\\_I2C\\_MASTER\\_HANDLER\\_SIZE](#)  
*ak4497 handle size*
- #define [AK4497\\_CONTROL1](#) (0x00U)  
*define the registers offset of AK4497.*
- #define [AK4497\\_CONTROL1\\_RSTN\\_MASK](#) (0x1U)  
*define BIT info of AK4497.*
- #define [AK4497\\_I2C\\_ADDR](#) (0x11U)  
*AK4497 I2C address.*
- #define [AK4497\\_I2C\\_BITRATE](#) (1000000U)  
*AK4497 i2c baudrate.*

### Enumerations

- enum [ak4497\\_mode\\_t](#)  
*The AK4497 playback mode.*
- enum [ak4497\\_data\\_channel\\_mode\\_t](#) {  
  [kAK4497\\_NormalMode](#) = 0x0,  
  [kAK4497\\_ExchangeMode](#) = 0x1 }  
*The Data selection of L-channel and R-channel for DSD mode, defined by SELLR bit.*
- enum [ak4497\\_dsd\\_input\\_path\\_t](#) {  
  [kAK4497\\_Path0](#) = 0x0,  
  [kAK4497\\_Path1](#) = 0x1 }  
*The data path select for DSD mode.*
- enum [ak4497\\_dsd\\_mclk\\_t](#) {  
  [kAK4497\\_mclk512fs](#) = 0x0,  
  [kAK4497\\_mclk768fs](#) = 0x1 }  
*The MCLK select for DSD mode, defined by DCKS bit.*

- enum `ak4497_dsd_dclk_t` {  
`kAK4497_dclk64fs` = 0x0,  
`kAK4497_dclk128fs` = 0x1,  
`kAK4497_dclk256fs` = 0x2,  
`kAK4497_dclk512fs` = 0x3 }  
*The DCLK select for DSD mode, defined by DSDSEL[1:0].*
- enum `ak4497_dsd_playback_path_t` {  
`kAK4497_NormalPath` = 0x0,  
`kAK4497_VolumeBypass` = 0x1 }  
*DSD playback path.*
- enum `ak4497_dsd_data_mute_t`  
*DSD mute flag.*
- enum `ak4497_dsd_dclk_polarity_t` {  
`kAK4497_FallingEdge` = 0x0,  
`kAK4497_RisingEdge` = 0x1 }  
*DSD bclk polarity.*
- enum `ak4497_pcm_samplefreqmode_t` {  
`kAK4497_ManualSettingMode` = 0x0,  
`kAK4497_AutoSettingMode` = 0x1,  
`kAK4497_FsAutoDetectMode` = 0x2 }  
*The sampling frequency mode for PCM and EXDF mode, defined by CR01[AFSD], CR00[ACKS].*
- enum `ak4497_pcm_samplefreqselect_t` {  
`kAK4497_NormalSpeed` = 0x0,  
`kAK4497_DoubleSpeed` = 0x1,  
`kAK4497_QuadSpeed` = 0x2,  
`kAK4497_OctSpeed` = 0x4,  
`kAK4497_HexSpeed` = 0x5 }  
*The sampling speed select, defined by DFS[2:0].*
- enum `ak4497_pcm_sdata_format_t` {  
`kAK4497_16BitLSB` = 0x0,  
`kAK4497_20BitLSB` = 0x1,  
`kAK4497_24BitMSB` = 0x2,  
`kAK4497_16_24BitI2S` = 0x3,  
`kAK4497_24BitLSB` = 0x4,  
`kAK4497_32BitLSB` = 0x5,  
`kAK4497_32BitMSB` = 0x6,  
`kAK4497_32BitI2S` = 0x7 }  
*The audio data interface modes, defined by DIF[2:0].*
- enum `ak4497_pcm_tdm_mode_t` {  
`kAK4497_Normal` = 0x0,  
`kAK4497_TDM128` = 0x1,  
`kAK4497_TDM256` = 0x2,  
`kAK4497_TDM512` = 0x3 }  
*The TDM mode select, defined by TDM[1:0].*
- enum `ak4497_pcm_sds_select_t`  
*The audio data slot selection, defined by SDS[2:0].*
- enum `ak4497_module_ctrl_cmd_t` { `kAK4497_ModuleSwitchI2SInInterface` = 0U }

- *audio codec module control cmd*
- enum {  
     kAK4497\_ModuleI2SInInterfacePCM = 0U,  
     kAK4497\_ModuleI2SInInterfaceDSD = 1U }  
     *audio codec module digital interface*

## Functions

- void AK4497\_DefaultConfig (ak4497\_config\_t \*config)  
     *Default initializes AK4497.*
- status\_t AK4497\_Init (ak4497\_handle\_t \*handle, ak4497\_config\_t \*config)  
     *Initializes AK4497.*
- status\_t AK4497\_SetEncoding (ak4497\_handle\_t \*handle, uint8\_t format)  
     *Set the codec PCM mode or DSD mode based on the format info.*
- status\_t AK4497\_ConfigDataFormat (ak4497\_handle\_t \*handle, uint32\_t mclk, uint32\_t sample-Rate, uint32\_t bitWidth)  
     *Configure the data format of audio data.*
- status\_t AK4497\_SetVolume (ak4497\_handle\_t \*handle, uint8\_t value)  
     *Set the volume of different modules in AK4497.*
- status\_t AK4497\_GetVolume (ak4497\_handle\_t \*handle, uint8\_t \*value)  
     *Get the volume of different modules in AK4497.*
- status\_t AK4497\_ModuleControl (ak4497\_handle\_t \*handle, ak4497\_module\_ctrl\_cmd\_t cmd, uint32\_t data)  
     *AK4497 codec module control.*
- status\_t AK4497\_Deinit (ak4497\_handle\_t \*handle)  
     *Deinit the AK4497 codec.*
- status\_t AK4497\_WriteReg (ak4497\_handle\_t \*handle, uint8\_t reg, uint8\_t val)  
     *Write register to AK4497 using I2C.*
- status\_t AK4497\_ReadReg (ak4497\_handle\_t \*handle, uint8\_t reg, uint8\_t \*val)  
     *Read register from AK4497 using I2C.*
- status\_t AK4497\_ModifyReg (ak4497\_handle\_t \*handle, uint8\_t reg, uint8\_t mask, uint8\_t val)  
     *Modify some bits in the register using I2C.*

## Driver version

- #define FSL\_AK4497\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))  
     *CLOCK driver version 2.1.2.*



## 25.4.2 Data Structure Documentation

### 25.4.2.1 struct ak4497\_dsd\_config\_t

### 25.4.2.2 struct ak4497\_pcm\_config\_t

### 25.4.2.3 struct ak4497\_config\_t

#### Data Fields

- uint8\_t [slaveAddress](#)  
*code device slave address*
- [codec\\_i2c\\_config\\_t](#) [i2cConfig](#)  
*i2c bus configuration*

### 25.4.2.4 struct ak4497\_handle\_t

#### Data Fields

- [ak4497\\_config\\_t](#) \* [config](#)  
*ak4497 config pointer*
- uint8\_t [i2cHandle](#) [[AK4497\\_I2C\\_HANDLER\\_SIZE](#)]  
*i2c handle*

## 25.4.3 Macro Definition Documentation

### 25.4.3.1 #define AK4497\_CONTROL1 (0x00U)

### 25.4.3.2 #define AK4497\_CONTROL1\_RSTN\_MASK (0x1U)

### 25.4.3.3 #define AK4497\_I2C\_ADDR (0x11U)

## 25.4.4 Enumeration Type Documentation

### 25.4.4.1 enum ak4497\_data\_channel\_mode\_t

#### Enumerator

***kAK4497\_NormalMode*** L-channel output L-channel data, R-channel output R-channel data.

***kAK4497\_ExchangeMode*** L-channel output R-channel data, R-channel output L-channel data.

**25.4.4.2 enum ak4497\_dsd\_input\_path\_t**

Enumerator

*kAK4497\_Path0* Pin 16,17,19 used.*kAK4497\_Path1* Pin 3,4,5 used.**25.4.4.3 enum ak4497\_dsd\_mclk\_t**

Enumerator

*kAK4497\_mclk512fs* MCLK equals 512fs.*kAK4497\_mclk768fs* MCLK equals 768fs.**25.4.4.4 enum ak4497\_dsd\_dclk\_t**

Enumerator

*kAK4497\_dclk64fs* DCLK equals 64fs.*kAK4497\_dclk128fs* DCLK equals 128fs.*kAK4497\_dclk256fs* DCLK equals 256fs.*kAK4497\_dclk512fs* DCLK equals 512fs.**25.4.4.5 enum ak4497\_dsd\_playback\_path\_t**

Enumerator

*kAK4497\_NormalPath* Normal path mode.*kAK4497\_VolumeBypass* Volume Bypass mode.**25.4.4.6 enum ak4497\_dsd\_dclk\_polarity\_t**

Enumerator

*kAK4497\_FallingEdge* DSD data is output from DCLK falling edge.*kAK4497\_RisingEdge* DSD data is output from DCLK rising edge.**25.4.4.7 enum ak4497\_pcm\_samplefreqmode\_t**

Enumerator

*kAK4497\_ManualSettingMode* Manual setting mode.*kAK4497\_AutoSettingMode* Auto setting mode.*kAK4497\_FsAutoDetectMode* Auto detect mode.

#### 25.4.4.8 enum ak4497\_pcm\_samplefreqselect\_t

Enumerator

*kAK4497\_NormalSpeed* 8kHz ~ 54kHz  
*kAK4497\_DoubleSpeed* 54kHz ~ 108kHz  
*kAK4497\_QuadSpeed* 120kHz ~ 216kHz, note that value 3 also stands for Quad Speed Mode  
*kAK4497\_OctSpeed* 384kHz, note that value 6 also stands for Oct Speed Mode  
*kAK4497\_HexSpeed* 768kHz, note that value 7 also stands for Hex Speed Mode

#### 25.4.4.9 enum ak4497\_pcm\_sdata\_format\_t

Enumerator

*kAK4497\_16BitLSB* 16-bit LSB justified  
*kAK4497\_20BitLSB* 20-bit LSB justified  
*kAK4497\_24BitMSB* 24-bit MSB justified  
*kAK4497\_16\_24BitI2S* 16 and 24-bit I2S compatible  
*kAK4497\_24BitLSB* 24-bit LSB justified  
*kAK4497\_32BitLSB* 32-bit LSB justified  
*kAK4497\_32BitMSB* 32-bit MSB justified  
*kAK4497\_32BitI2S* 32-bit I2S compatible

#### 25.4.4.10 enum ak4497\_pcm\_tdm\_mode\_t

Enumerator

*kAK4497\_Normal* Normal mode.  
*kAK4497\_TDM128* BCLK is fixed to 128fs.  
*kAK4497\_TDM256* BCLK is fixed to 256fs.  
*kAK4497\_TDM512* BCLK is fixed to 512fs.

#### 25.4.4.11 enum ak4497\_module\_ctrl\_cmd\_t

Enumerator

*kAK4497\_ModuleSwitchI2SInInterface* module digital interface switch.

#### 25.4.4.12 anonymous enum

Enumerator

*kAK4497\_ModuleI2SInInterfacePCM* Pcm interface.  
*kAK4497\_ModuleI2SInInterfaceDSD* DSD interface.

## 25.4.5 Function Documentation

25.4.5.1 void AK4497\_DefaultConfig ( ak4497\_config\_t \* *config* )

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>config</i> | AK4497 configure structure. |
|---------------|-----------------------------|

**25.4.5.2** `status_t AK4497_Init ( ak4497_handle_t * handle, ak4497_config_t * config )`

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>handle</i> | AK4497 handle structure.    |
| <i>config</i> | AK4497 configure structure. |

**25.4.5.3** `status_t AK4497_SetEncoding ( ak4497_handle_t * handle, uint8_t format )`

This function would configure the codec playback mode.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | AK4497 handle structure pointer. |
| <i>format</i> | info.                            |

**25.4.5.4** `status_t AK4497_ConfigDataFormat ( ak4497_handle_t * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

This function would configure the registers about the sample rate, bit depths.

Parameters

|                   |                                                                         |
|-------------------|-------------------------------------------------------------------------|
| <i>handle</i>     | AK4497 handle structure pointer.                                        |
| <i>mclk</i>       | system clock of the codec which can be generated by MCLK or PLL output. |
| <i>sampleRate</i> | Sample rate of audio file running in AK4497.                            |
| <i>bitWidth</i>   | Bit depth of audio file.                                                |

**25.4.5.5** `status_t AK4497_SetVolume ( ak4497_handle_t * handle, uint8_t value )`

This function would set the volume of AK4497 modules. Users need to appoint the module. The function assume that left channel and right channel has the same volume.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>handle</i> | AK4497 handle structure.     |
| <i>value</i>  | Volume value need to be set. |

**25.4.5.6 status\_t AK4497\_GetVolume ( ak4497\_handle\_t \* *handle*, uint8\_t \* *value* )**

This function gets the volume of AK4497. Users need to appoint the module. The function assume that left channel and right channel has the same volume.

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | AK4497 handle structure. |
| <i>value</i>  | volume value             |

## Returns

value value of the module.

**25.4.5.7 status\_t AK4497\_ModuleControl ( ak4497\_handle\_t \* *handle*, ak4497\_module\_ctrl\_cmd\_t *cmd*, uint32\_t *data* )**

## Parameters

|               |                                                                                                |
|---------------|------------------------------------------------------------------------------------------------|
| <i>handle</i> | AK4497 handle structure pointer.                                                               |
| <i>cmd</i>    | module control command, support cmd kAK4497_ModuleSwitchDigitalInterface.                      |
| <i>data</i>   | control data, support data kCODEC_ModuleDigitalInterfacePCM/kCODEC_-ModuleDigitalInterfaceDSD. |

**25.4.5.8 status\_t AK4497\_Deinit ( ak4497\_handle\_t \* *handle* )**

This function close all modules in AK4497 to save power.

## Parameters

---

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | AK4497 handle structure pointer. |
|---------------|----------------------------------|

#### 25.4.5.9 **status\_t AK4497\_WriteReg ( ak4497\_handle\_t \* *handle*, uint8\_t *reg*, uint8\_t *val* )**

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>handle</i> | AK4497 handle structure.                |
| <i>reg</i>    | The register address in AK4497.         |
| <i>val</i>    | Value needs to write into the register. |

#### 25.4.5.10 **status\_t AK4497\_ReadReg ( ak4497\_handle\_t \* *handle*, uint8\_t *reg*, uint8\_t \* *val* )**

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>handle</i> | AK4497 handle structure.        |
| <i>reg</i>    | The register address in AK4497. |
| <i>val</i>    | Value written to.               |

#### 25.4.5.11 **status\_t AK4497\_ModifyReg ( ak4497\_handle\_t \* *handle*, uint8\_t *reg*, uint8\_t *mask*, uint8\_t *val* )**

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>handle</i> | AK4497 handle structure.                                                         |
| <i>reg</i>    | The register address in AK4497.                                                  |
| <i>mask</i>   | The mask code for the bits want to write. The bit you want to write should be 0. |
| <i>val</i>    | Value needs to write into the register.                                          |

## 25.4.6 AK4497 Adapter

### 25.4.6.1 Overview

The ak4497 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_AK4497_HANDLER_SIZE` (`AK4497_I2C_HANDLER_SIZE` + 4)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_AK4497_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_AK4497_Deinit` (void \*handle)  
*Codec de-initialization.*
- `status_t HAL_CODEC_AK4497_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_AK4497_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_AK4497_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_AK4497_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_AK4497_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_AK4497_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_AK4497_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_AK4497_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initialization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initialization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)



- *set audio codec module power.*  
static [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)
- *codec set record source.*  
static [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)
- *codec set record channel.*  
static [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)
- *codec set play source.*  
static [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)
- *codec module control.*

## 25.4.6.2 Function Documentation

### 25.4.6.2.1 status\_t HAL\_CODEC\_AK4497\_Init ( void \* *handle*, void \* *config* )

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

### 25.4.6.2.2 status\_t HAL\_CODEC\_AK4497\_Deinit ( void \* *handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 25.4.6.2.3 status\_t HAL\_CODEC\_AK4497\_SetFormat ( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* )

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.4.6.2.4 status\_t HAL\_CODEC\_AK4497\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.4.6.2.5 status\_t HAL\_CODEC\_AK4497\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.4.6.2.6 status\_t HAL\_CODEC\_AK4497\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.4.6.2.7 status\_t HAL\_CODEC\_AK4497\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.4.6.2.8 status\_t HAL\_CODEC\_AK4497\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

## Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.4.6.2.9 status\_t HAL\_CODEC\_AK4497\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 25.4.6.2.10 `status_t HAL_CODEC_AK4497_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

## Returns

`kStatus_Success` is success, else configure failed.

#### 25.4.6.2.11 `static status_t HAL_CODEC_Init ( void * handle, void * config ) [inline], [static]`

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

## Returns

`kStatus_Success` is success, else initial failed.

#### 25.4.6.2.12 `static status_t HAL_CODEC_Deinit ( void * handle ) [inline], [static]`

## Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## Returns

kStatus\_Success is success, else de-initial failed.

**25.4.6.2.13** `static status_t HAL_CODEC_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth ) [inline], [static]`

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

**25.4.6.2.14** `static status_t HAL_CODEC_SetVolume ( void * handle, uint32_t playChannel, uint32_t volume ) [inline], [static]`

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

**25.4.6.2.15** `static status_t HAL_CODEC_SetMute ( void * handle, uint32_t playChannel, bool isMute ) [inline], [static]`

## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

## Returns

kStatus\_Success is success, else configure failed.

**25.4.6.2.16** `static status_t HAL_CODEC_SetPower ( void * handle, uint32_t module, bool powerOn ) [inline], [static]`

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

kStatus\_Success is success, else configure failed.

**25.4.6.2.17** `static status_t HAL_CODEC_SetRecord ( void * handle, uint32_t recordSource ) [inline], [static]`

## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

kStatus\_Success is success, else configure failed.

**25.4.6.2.18** `static status_t HAL_CODEC_SetRecordChannel ( void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel ) [inline], [static]`

## Parameters

|                            |                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                                                |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine of member in <code>_codec_record_channel</code> .       |

## Returns

`kStatus_Success` is success, else configure failed.

**25.4.6.2.19** `static status_t HAL_CODEC_SetPlay ( void * handle, uint32_t playSource )`  
**[inline], [static]**

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**25.4.6.2.20** `static status_t HAL_CODEC_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`  
**[inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

## Returns

`kStatus_Success` is success, else configure failed.

## 25.5 WM8524 Driver

### 25.5.1 Overview

The wm8524 driver provides a codec control interface.

### Data Structures

- struct `wm8524_handle_t`  
WM8524 handler. *More...*

### Typedefs

- typedef void(\* `wm8524_setMuteIO`)(uint32\_t output)  
< mute control io function pointer

### Enumerations

- enum `wm8524_protocol_t` {  
    `kWM8524_ProtocolLeftJustified` = 0x0,  
    `kWM8524_ProtocolI2S` = 0x1,  
    `kWM8524_ProtocolRightJustified` = 0x2 }  
    *The audio data transfer protocol.*
- enum `_wm8524_mute_control` {  
    `kWM8524_Mute` = 0U,  
    `kWM8524_Unmute` = 1U }  
    *wm8524 mute operation*

### Functions

- `status_t WM8524_Init`(`wm8524_handle_t` \*handle, `wm8524_config_t` \*config)  
    *Initializes WM8524.*
- void `WM8524_ConfigFormat`(`wm8524_handle_t` \*handle, `wm8524_protocol_t` protocol)  
    *Configure WM8524 audio protocol.*
- void `WM8524_SetMute`(`wm8524_handle_t` \*handle, bool isMute)  
    *Sets the codec mute state.*

### Driver version

- #define `FSL_WM8524_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 1))  
    *WM8524 driver version 2.1.1.*



## 25.5.2 Data Structure Documentation

### 25.5.2.1 struct wm8524\_handle\_t

#### Data Fields

- wm8524\_config\_t \* [config](#)  
*wm8524 config pointer*

## 25.5.3 Macro Definition Documentation

### 25.5.3.1 #define FSL\_WM8524\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

## 25.5.4 Typedef Documentation

### 25.5.4.1 typedef void(\* wm8524\_setMutelO)(uint32\_t output)

format control io function pointer

## 25.5.5 Enumeration Type Documentation

### 25.5.5.1 enum wm8524\_protocol\_t

Enumerator

*kWM8524\_ProtocolLeftJustified* Left justified mode.  
*kWM8524\_ProtocolI2S* I2S mode.  
*kWM8524\_ProtocolRightJustified* Right justified mode.

### 25.5.5.2 enum \_wm8524\_mute\_control

Enumerator

*kWM8524\_Mute* mute left and right channel DAC  
*kWM8524\_Unmute* unmute left and right channel DAC

## 25.5.6 Function Documentation

### 25.5.6.1 status\_t WM8524\_Init ( wm8524\_handle\_t \* *handle*, wm8524\_config\_t \* *config* )

## Parameters

|               |                             |
|---------------|-----------------------------|
| <i>handle</i> | WM8524 handle structure.    |
| <i>config</i> | WM8524 configure structure. |

## Returns

kStatus\_Success.

### 25.5.6.2 void WM8524\_ConfigFormat ( wm8524\_handle\_t \* *handle*, wm8524\_protocol\_t *protocol* )

## Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>handle</i>   | WM8524 handle structure.        |
| <i>protocol</i> | WM8524 configuration structure. |

### 25.5.6.3 void WM8524\_SetMute ( wm8524\_handle\_t \* *handle*, bool *isMute* )

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>handle</i> | WM8524 handle structure.             |
| <i>isMute</i> | true means mute, false means normal. |

## 25.5.7 WM8524 Adapter

### 25.5.7.1 Overview

The wm8524 adapter provides a codec unify control interface.

#### Macros

- #define [HAL\\_CODEC\\_WM8524\\_HANDLER\\_SIZE](#) (4)  
*codec handler size*

#### Functions

- [status\\_t HAL\\_CODEC\\_WM8524\\_Init](#) (void \*handle, void \*config)  
*Codec initialization.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_Deinit](#) (void \*handle)  
*Codec de-initialization.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_SetFormat](#) (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_SetVolume](#) (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_SetMute](#) (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_SetPower](#) (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_SetRecord](#) (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_SetPlay](#) (void \*handle, uint32\_t playSource)  
*codec set play source.*
- [status\\_t HAL\\_CODEC\\_WM8524\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static [status\\_t HAL\\_CODEC\\_Init](#) (void \*handle, void \*config)  
*Codec initialization.*
- static [status\\_t HAL\\_CODEC\\_Deinit](#) (void \*handle)  
*Codec de-initialization.*
- static [status\\_t HAL\\_CODEC\\_SetFormat](#) (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static [status\\_t HAL\\_CODEC\\_SetVolume](#) (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static [status\\_t HAL\\_CODEC\\_SetMute](#) (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static [status\\_t HAL\\_CODEC\\_SetPower](#) (void \*handle, uint32\_t module, bool powerOn)

- set audio codec module power.*
- static [status\\_t HAL\\_CODEC\\_SetRecord](#) (void \*handle, uint32\_t recordSource)
- codec set record source.*
- static [status\\_t HAL\\_CODEC\\_SetRecordChannel](#) (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)
- codec set record channel.*
- static [status\\_t HAL\\_CODEC\\_SetPlay](#) (void \*handle, uint32\_t playSource)
- codec set play source.*
- static [status\\_t HAL\\_CODEC\\_ModuleControl](#) (void \*handle, uint32\_t cmd, uint32\_t data)
- codec module control.*

## 25.5.7.2 Function Documentation

### 25.5.7.2.1 status\_t HAL\_CODEC\_WM8524\_Init ( void \* *handle*, void \* *config* )

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

### 25.5.7.2.2 status\_t HAL\_CODEC\_WM8524\_Deinit ( void \* *handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 25.5.7.2.3 status\_t HAL\_CODEC\_WM8524\_SetFormat ( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* )

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.5.7.2.4 status\_t HAL\_CODEC\_WM8524\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.5.7.2.5 status\_t HAL\_CODEC\_WM8524\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.5.7.2.6 status\_t HAL\_CODEC\_WM8524\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.5.7.2.7 status\_t HAL\_CODEC\_WM8524\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

## Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.5.7.2.8 status\_t HAL\_CODEC\_WM8524\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

## Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

## Returns

kStatus\_Success is success, else configure failed.

#### 25.5.7.2.9 status\_t HAL\_CODEC\_WM8524\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

#### 25.5.7.2.10 `status_t HAL_CODEC_WM8524_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

## Returns

`kStatus_Success` is success, else configure failed.

#### 25.5.7.2.11 `static status_t HAL_CODEC_Init ( void * handle, void * config ) [inline], [static]`

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

## Returns

`kStatus_Success` is success, else initial failed.

#### 25.5.7.2.12 `static status_t HAL_CODEC_Deinit ( void * handle ) [inline], [static]`

## Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

## Returns

kStatus\_Success is success, else de-initial failed.

**25.5.7.2.13** `static status_t HAL_CODEC_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth ) [inline], [static]`

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

## Returns

kStatus\_Success is success, else configure failed.

**25.5.7.2.14** `static status_t HAL_CODEC_SetVolume ( void * handle, uint32_t playChannel, uint32_t volume ) [inline], [static]`

## Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

## Returns

kStatus\_Success is success, else configure failed.

**25.5.7.2.15** `static status_t HAL_CODEC_SetMute ( void * handle, uint32_t playChannel, bool isMute ) [inline], [static]`



## Parameters

|                    |                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                                   |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> . |
| <i>isMute</i>      | true is mute, false is unmute.                                                                  |

## Returns

`kStatus_Success` is success, else configure failed.

**25.5.7.2.16** `static status_t HAL_CODEC_SetPower ( void * handle, uint32_t module, bool powerOn ) [inline], [static]`

## Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

## Returns

`kStatus_Success` is success, else configure failed.

**25.5.7.2.17** `static status_t HAL_CODEC_SetRecord ( void * handle, uint32_t recordSource ) [inline], [static]`

## Parameters

|                     |                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                                     |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of <code>_codec_record_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**25.5.7.2.18** `static status_t HAL_CODEC_SetRecordChannel ( void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel ) [inline], [static]`

## Parameters

|                            |                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                                                |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> . |
| <i>rightRecord-Channel</i> | audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine of member in <code>_codec_record_channel</code> .       |

## Returns

`kStatus_Success` is success, else configure failed.

**25.5.7.2.19** `static status_t HAL_CODEC_SetPlay ( void * handle, uint32_t playSource )`  
**[inline], [static]**

## Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                                 |
| <i>playSource</i> | audio codec play source, can be a value or combine value of <code>_codec_play_source</code> . |

## Returns

`kStatus_Success` is success, else configure failed.

**25.5.7.2.20** `static status_t HAL_CODEC_ModuleControl ( void * handle, uint32_t cmd, uint32_t data )`  
**[inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

## Parameters

|               |                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                                                     |
| <i>cmd</i>    | module control cmd, reference <code>_codec_module_ctrl_cmd</code> .                                                                                                                                                                                                                               |
| <i>data</i>   | value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations. |

## Returns

`kStatus_Success` is success, else configure failed.

# Chapter 26

## Serial Manager

### 26.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

### Modules

- [Serial Port SWO](#)
- [Serial Port Uart](#)

### Data Structures

- struct [serial\\_manager\\_config\\_t](#)  
*serial manager config structure [More...](#)*
- struct [serial\\_manager\\_callback\\_message\\_t](#)  
*Callback message structure. [More...](#)*

### Macros

- #define [SERIAL\\_MANAGER\\_NON\\_BLOCKING\\_MODE](#) (1U)  
*Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)*
- #define [SERIAL\\_MANAGER\\_RING\\_BUFFER\\_FLOWCONTROL](#) (0U)  
*Enable or ring buffer flow control (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_UART](#) (0U)  
*Enable or disable uart port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_UART\\_DMA](#) (0U)  
*Enable or disable uart dma port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_USBCDC](#) (0U)  
*Enable or disable USB CDC port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SWO](#) (0U)  
*Enable or disable SWO port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_VIRTUAL](#) (0U)  
*Enable or disable USB CDC virtual port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_RPMSG](#) (0U)  
*Enable or disable rPMSG port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SPI\\_MASTER](#) (0U)  
*Enable or disable SPI Master port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SPI\\_SLAVE](#) (0U)  
*Enable or disable SPI Slave port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_BLE\\_WU](#) (0U)  
*Enable or disable BLE WU port (1 - enable, 0 - disable)*
- #define [SERIAL\\_MANAGER\\_TASK\\_HANDLE\\_TX](#) (0U)

- *Enable or disable SerialManager\_Task() handle TX to prevent recursive calling.*  
• #define `SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE` (1U)  
*Set the default delay time in ms used by SerialManager\_WriteTimeDelay().*
- #define `SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE` (1U)  
*Set the default delay time in ms used by SerialManager\_ReadTimeDelay().*
- #define `SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY` (0U)  
*Enable or disable SerialManager\_Task() handle RX data available notify.*
- #define `SERIAL_MANAGER_WRITE_HANDLE_SIZE` (44U)  
*Set serial manager write handle size.*
- #define `SERIAL_MANAGER_USE_COMMON_TASK` (0U)  
*SERIAL\_PORT\_UART\_HANDLE\_SIZE/SERIAL\_PORT\_USB\_CDC\_HANDLE\_SIZE + serial manager dedicated size.*
- #define `SERIAL_MANAGER_HANDLE_SIZE` (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 124U)  
*Definition of serial manager handle size.*
- #define `SERIAL_MANAGER_HANDLE_DEFINE`(name) uint32\_t name[(((`SERIAL_MANAGER_HANDLE_SIZE` + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]  
*Defines the serial manager handle.*
- #define `SERIAL_MANAGER_WRITE_HANDLE_DEFINE`(name) uint32\_t name[(((`SERIAL_MANAGER_WRITE_HANDLE_SIZE` + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]  
*Defines the serial manager write handle.*
- #define `SERIAL_MANAGER_READ_HANDLE_DEFINE`(name) uint32\_t name[(((`SERIAL_MANAGER_READ_HANDLE_SIZE` + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]  
*Defines the serial manager read handle.*
- #define `SERIAL_MANAGER_TASK_PRIORITY` (2U)  
*Macro to set serial manager task priority.*
- #define `SERIAL_MANAGER_TASK_STACK_SIZE` (1000U)  
*Macro to set serial manager task stack size.*

## Typedefs

- typedef void \* `serial_handle_t`  
*The handle of the serial manager module.*
- typedef void \* `serial_write_handle_t`  
*The write handle of the serial manager module.*
- typedef void \* `serial_read_handle_t`  
*The read handle of the serial manager module.*
- typedef void(\* `serial_manager_callback_t` )(void \*callbackParam, `serial_manager_callback_message_t` \*message, `serial_manager_status_t` status)  
*serial manager callback function*
- typedef int32\_t(\* `serial_manager_lowpower_critical_callback_t` )(int32\_t power\_mode)  
*serial manager Lowpower Critical callback function*

## Enumerations

- enum `serial_port_type_t` {  
`kSerialPort_None` = 0U,  
`kSerialPort_Uart` = 1U,  
`kSerialPort_UsbCdc`,  
`kSerialPort_Swo`,  
`kSerialPort_Virtual`,  
`kSerialPort_Rpmsg`,  
`kSerialPort_UartDma`,  
`kSerialPort_SpiMaster`,  
`kSerialPort_SpiSlave`,  
`kSerialPort_BleWu` }  
*serial port type*
- enum `serial_manager_type_t` {  
`kSerialManager_NonBlocking` = 0x0U,  
`kSerialManager_Blocking` = 0x8F41U }  
*serial manager type*
- enum `serial_manager_status_t` {  
`kStatus_SerialManager_Success` = `kStatus_Success`,  
`kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,  
`kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,  
`kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,  
`kStatus_SerialManager_Canceled`,  
`kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,  
`kStatus_SerialManager_RingBufferOverflow`,  
`kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }  
*serial manager error code*

## Functions

- `serial_manager_status_t SerialManager_Init` (`serial_handle_t` serialHandle, const `serial_manager_config_t` \*config)  
*Initializes a serial manager module with the serial manager handle and the user configuration structure.*
- `serial_manager_status_t SerialManager_Deinit` (`serial_handle_t` serialHandle)  
*De-initializes the serial manager module instance.*
- `serial_manager_status_t SerialManager_OpenWriteHandle` (`serial_handle_t` serialHandle, `serial_write_handle_t` writeHandle)  
*Opens a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseWriteHandle` (`serial_write_handle_t` writeHandle)  
*Closes a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_OpenReadHandle` (`serial_handle_t` serialHandle, `serial_read_handle_t` readHandle)  
*Opens a reading handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseReadHandle` (`serial_read_handle_t` readHandle)  
*Closes a reading for the serial manager module.*

- `serial_manager_status_t SerialManager_WriteBlocking` (`serial_write_handle_t` writeHandle, `uint8_t` \*buffer, `uint32_t` length)  
*Transmits data with the blocking mode.*
- `serial_manager_status_t SerialManager_ReadBlocking` (`serial_read_handle_t` readHandle, `uint8_t` \*buffer, `uint32_t` length)  
*Reads data with the blocking mode.*
- `serial_manager_status_t SerialManager_WriteNonBlocking` (`serial_write_handle_t` writeHandle, `uint8_t` \*buffer, `uint32_t` length)  
*Transmits data with the non-blocking mode.*
- `serial_manager_status_t SerialManager_ReadNonBlocking` (`serial_read_handle_t` readHandle, `uint8_t` \*buffer, `uint32_t` length)  
*Reads data with the non-blocking mode.*
- `serial_manager_status_t SerialManager_TryRead` (`serial_read_handle_t` readHandle, `uint8_t` \*buffer, `uint32_t` length, `uint32_t` \*receivedLength)  
*Tries to read data.*
- `serial_manager_status_t SerialManager_CancelWriting` (`serial_write_handle_t` writeHandle)  
*Cancels unfinished send transmission.*
- `serial_manager_status_t SerialManager_CancelReading` (`serial_read_handle_t` readHandle)  
*Cancels unfinished receive transmission.*
- `serial_manager_status_t SerialManager_InstallTxCallback` (`serial_write_handle_t` writeHandle, `serial_manager_callback_t` callback, void \*callbackParam)  
*Installs a TX callback and callback parameter.*
- `serial_manager_status_t SerialManager_InstallRxCallback` (`serial_read_handle_t` readHandle, `serial_manager_callback_t` callback, void \*callbackParam)  
*Installs a RX callback and callback parameter.*
- static bool `SerialManager_needPollingIsr` (void)  
*Check if need polling ISR.*
- `serial_manager_status_t SerialManager_EnterLowpower` (`serial_handle_t` serialHandle)  
*Prepares to enter low power consumption.*
- `serial_manager_status_t SerialManager_ExitLowpower` (`serial_handle_t` serialHandle)  
*Restores from low power consumption.*
- void `SerialManager_SetLowpowerCriticalCb` (const `serial_manager_lowpower_critical_CBs_t` \*pfCallback)  
*This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.*

## 26.2 Data Structure Documentation

### 26.2.1 struct serial\_manager\_config\_t

#### Data Fields

- `uint8_t` \* `ringBuffer`  
*Ring buffer address, it is used to buffer data received by the hardware.*
- `uint32_t` `ringBufferSize`  
*The size of the ring buffer.*
- `serial_port_type_t` type  
*Serial port type.*
- `serial_manager_type_t` blockType

- *Serial manager port type.*  
void \* `portConfig`  
*Serial port configuration.*

## Field Documentation

### (1) uint8\_t\* serial\_manager\_config\_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

## 26.2.2 struct serial\_manager\_callback\_message\_t

### Data Fields

- uint8\_t \* `buffer`  
*Transferred buffer.*
- uint32\_t `length`  
*Transferred data length.*

## 26.3 Macro Definition Documentation

### 26.3.1 #define SERIAL\_MANAGER\_WRITE\_TIME\_DELAY\_DEFAULT\_VALUE (1U)

### 26.3.2 #define SERIAL\_MANAGER\_READ\_TIME\_DELAY\_DEFAULT\_VALUE (1U)

### 26.3.3 #define SERIAL\_MANAGER\_USE\_COMMON\_TASK (0U)

Macro to determine whether use common task.

### 26.3.4 #define SERIAL\_MANAGER\_HANDLE\_SIZE (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 124U)

### 26.3.5 #define SERIAL\_MANAGER\_HANDLE\_DEFINE( *name* ) uint32\_t name[(((SERIAL\_MANAGER\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)))]

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial\_handle\_t)name" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);
*
```

Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>name</i> | The name string of the serial manager handle. |
|-------------|-----------------------------------------------|

### 26.3.6 #define SERIAL\_MANAGER\_WRITE\_HANDLE\_DEFINE( *name* ) uint32\_t name[(((SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial\_write\_handle-  
\_t)name" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);
*
```

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>name</i> | The name string of the serial manager write handle. |
|-------------|-----------------------------------------------------|

### 26.3.7 #define SERIAL\_MANAGER\_READ\_HANDLE\_DEFINE( *name* ) uint32\_t name[(((SERIAL\_MANAGER\_READ\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial\_read\_handle-  
\_t)name" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```



## Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>name</i> | The name string of the serial manager read handle. |
|-------------|----------------------------------------------------|

**26.3.8 #define SERIAL\_MANAGER\_TASK\_PRIORITY (2U)****26.3.9 #define SERIAL\_MANAGER\_TASK\_STACK\_SIZE (1000U)****26.4 Enumeration Type Documentation****26.4.1 enum serial\_port\_type\_t**

## Enumerator

*kSerialPort\_None* Serial port is none.  
*kSerialPort\_Uart* Serial port UART.  
*kSerialPort\_UsbCdc* Serial port USB CDC.  
*kSerialPort\_Swo* Serial port SWO.  
*kSerialPort\_Virtual* Serial port Virtual.  
*kSerialPort\_Rpmsg* Serial port RPMSG.  
*kSerialPort\_UartDma* Serial port UART DMA.  
*kSerialPort\_SpiMaster* Serial port SPIMASTER.  
*kSerialPort\_SpiSlave* Serial port SPISLAVE.  
*kSerialPort\_BleWu* Serial port BLE WU.

**26.4.2 enum serial\_manager\_type\_t**

## Enumerator

*kSerialManager\_NonBlocking* None blocking handle.  
*kSerialManager\_Blocking* Blocking handle.

**26.4.3 enum serial\_manager\_status\_t**

## Enumerator

*kStatus\_SerialManager\_Success* Success.  
*kStatus\_SerialManager\_Error* Failed.  
*kStatus\_SerialManager\_Busy* Busy.  
*kStatus\_SerialManager\_Notify* Ring buffer is not empty.  
*kStatus\_SerialManager\_Canceled* the non-blocking request is canceled

***kStatus\_SerialManager\_HandleConflict*** The handle is opened.

***kStatus\_SerialManager\_RingBufferOverflow*** The ring buffer is overflowed.

***kStatus\_SerialManager\_NotConnected*** The host is not connected.

## 26.5 Function Documentation

### 26.5.1 `serial_manager_status_t SerialManager_Init ( serial_handle_t serialHandle, const serial_manager_config_t * config )`

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter `serialHandle` is a pointer to point to a memory space of size [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to [serial\\_port\\_type\\_t](#) for serial port setting. These three types can be set by using [serial\\_manager\\_config\\_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
* uartConfig.clockRate = 24000000;
* uartConfig.baudRate = 115200;
* uartConfig.parityMode = kSerialManager_UartParityDisabled;
* uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
* uartConfig.enableRx = 1;
* uartConfig.enableTx = 1;
* uartConfig.enableRxRTS = 0;
* uartConfig.enableTxCTS = 0;
* config.portConfig = &uartConfig;
* SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

For USB CDC,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_usb_cdc_config_t usbCdcConfig;
* config.type = kSerialPort_UsbCdc;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* usbCdcConfig.controllerIndex = kSerialManager_UsbControllerKhci0;
* config.portConfig = &usbCdcConfig;
* SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | Pointer to point to a memory space of size <a href="#">SERIAL_MANAGER_HANDLE_SIZE</a> allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_HANDLE_DEFINE(serialHandle)</a> ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |
| <i>config</i>       | Pointer to user-defined configuration structure.                                                                                                                                                                                                                                                                                                                                                                                                       |

## Return values

|                                      |                                                   |
|--------------------------------------|---------------------------------------------------|
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                                |
| <i>kStatus_SerialManager_Success</i> | The Serial Manager module initialization succeed. |

### 26.5.2 serial\_manager\_status\_t SerialManager\_Deinit ( serial\_handle\_t serialHandle )

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return `kStatus_SerialManager_Busy`.

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                      |                                                 |
|--------------------------------------|-------------------------------------------------|
| <i>kStatus_SerialManager_Success</i> | The serial manager de-initialization succeed.   |
| <i>kStatus_SerialManager_Busy</i>    | Opened reading or writing handle is not closed. |

### 26.5.3 serial\_manager\_status\_t SerialManager\_OpenWriteHandle ( serial\_handle\_t serialHandle, serial\_write\_handle\_t writeHandle )

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling [SerialManager\\_OpenWriteHandle](#). Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.                                                                                                                                                                                                                                                       |
| <i>writeHandle</i>  | The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle)</a> ; or <code>uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |

## Return values

|                                             |                                |
|---------------------------------------------|--------------------------------|
| <i>kStatus_SerialManager_Error</i>          | An error occurred.             |
| <i>kStatus_SerialManager_HandleConflict</i> | The writing handle was opened. |
| <i>kStatus_SerialManager_Success</i>        | The writing handle is opened.  |

Example below shows how to use this API to write data. For task 1,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
* static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
* , (serial_write_handle_t)s_serialWriteHandle1);
* SerialManager_InstallTxCallback((
* serial_write_handle_t)s_serialWriteHandle1,
* Task1_SerialManagerTxCallback,
* s_serialWriteHandle1);
* SerialManager_WriteNonBlocking((
* serial_write_handle_t)s_serialWriteHandle1,
* s_nonBlockingWelcome1,
* sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
* static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
* , (serial_write_handle_t)s_serialWriteHandle2);
* SerialManager_InstallTxCallback((
* serial_write_handle_t)s_serialWriteHandle2,
* Task2_SerialManagerTxCallback,
* s_serialWriteHandle2);
* SerialManager_WriteNonBlocking((
* serial_write_handle_t)s_serialWriteHandle2,
* s_nonBlockingWelcome2,
* sizeof(s_nonBlockingWelcome2) - 1U);
*
```

#### 26.5.4 serial\_manager\_status\_t SerialManager\_CloseWriteHandle ( serial\_write\_handle\_t *writeHandle* )

This function Closes a writing handle for the serial manager module.

## Parameters

|                    |                                                   |
|--------------------|---------------------------------------------------|
| <i>writeHandle</i> | The serial manager module writing handle pointer. |
|--------------------|---------------------------------------------------|

## Return values

|                                      |                               |
|--------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_Success</i> | The writing handle is closed. |
|--------------------------------------|-------------------------------|

### 26.5.5 serial\_manager\_status\_t SerialManager\_OpenReadHandle ( serial\_handle\_t serialHandle, serial\_read\_handle\_t readHandle )

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code `kStatus_SerialManager_Busy` would be returned when the previous reading handle is not closed. And there can only be one buffer for receiving for the reading handle at the same time.

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.                                                                                                                                                                                                                                                      |
| <i>readHandle</i>   | The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle)</a> ; or <code>uint32_t readHandle[(((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t)))]</code> ; |

## Return values

|                                      |                                        |
|--------------------------------------|----------------------------------------|
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                     |
| <i>kStatus_SerialManager_Success</i> | The reading handle is opened.          |
| <i>kStatus_SerialManager_Busy</i>    | Previous reading handle is not closed. |

Example below shows how to use this API to read data.

```
* static SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
* SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
* (serial_read_handle_t)s_serialReadHandle);
* static uint8_t s_nonBlockingBuffer[64];
* SerialManager_InstallRxCallback((
* serial_read_handle_t)s_serialReadHandle,
* APP_SerialManagerRxCallback,
* s_serialReadHandle);
```

```

* SerialManager_ReadNonBlocking((
 serial_read_handle_t)s_serialReadHandle,
* s_nonBlockingBuffer,
* sizeof(s_nonBlockingBuffer));
*

```

### 26.5.6 serial\_manager\_status\_t SerialManager\_CloseReadHandle ( serial\_read\_handle\_t *readHandle* )

This function Closes a reading for the serial manager module.

Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>readHandle</i> | The serial manager module reading handle pointer. |
|-------------------|---------------------------------------------------|

Return values

|                                      |                               |
|--------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_Success</i> | The reading handle is closed. |
|--------------------------------------|-------------------------------|

### 26.5.7 serial\_manager\_status\_t SerialManager\_WriteBlocking ( serial\_write\_handle\_t *writeHandle*, uint8\_t \* *buffer*, uint32\_t *length* )

This is a blocking function, which polls the sending queue, waits for the sending queue to be empty. This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

Note

The function [SerialManager\\_WriteBlocking](#) and the function `SerialManager_WriteNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelWriting` cannot be used to abort the transmission of this function.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
|--------------------|-------------------------------------------|

|               |                                     |
|---------------|-------------------------------------|
| <i>buffer</i> | Start address of the data to write. |
| <i>length</i> | Length of the data to write.        |

Return values

|                                            |                                                                  |
|--------------------------------------------|------------------------------------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Successfully sent all data.                                      |
| <i>kStatus_SerialManager_-<br/>Busy</i>    | Previous transmission still not finished; data not all sent yet. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                                               |

### 26.5.8 serial\_manager\_status\_t SerialManager\_ReadBlocking ( serial\_read\_handle\_t readHandle, uint8\_t \* buffer, uint32\_t length )

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

Note

The function [SerialManager\\_ReadBlocking](#) and the function `SerialManager_ReadNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelReading` cannot be used to abort the transmission of this function.

Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer.             |
| <i>buffer</i>     | Start address of the data to store the received data. |
| <i>length</i>     | The length of the data to be received.                |

Return values

|                                            |                                 |
|--------------------------------------------|---------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Successfully received all data. |
|--------------------------------------------|---------------------------------|



|                                    |                                                                      |
|------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_Busy</i>  | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_Error</i> | An error occurred.                                                   |

### 26.5.9 serial\_manager\_status\_t SerialManager\_WriteNonBlocking ( serial\_write\_handle\_t writeHandle, uint8\_t \* buffer, uint32\_t length )

This is a non-blocking function, which returns directly without waiting for all data to be sent. When all data is sent, the module notifies the upper layer through a TX callback function and passes the status parameter [kStatus\\_SerialManager\\_Success](#). This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

#### Note

The function [SerialManager\\_WriteBlocking](#) and the function [SerialManager\\_WriteNonBlocking](#) cannot be used at the same time. And, the TX callback is mandatory before the function could be used.

#### Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
| <i>buffer</i>      | Start address of the data to write.       |
| <i>length</i>      | Length of the data to write.              |

#### Return values

|                                      |                                                                  |
|--------------------------------------|------------------------------------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Successfully sent all data.                                      |
| <i>kStatus_SerialManager_Busy</i>    | Previous transmission still not finished; data not all sent yet. |
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                                               |

### 26.5.10 serial\_manager\_status\_t SerialManager\_ReadNonBlocking ( serial\_read\_handle\_t readHandle, uint8\_t \* buffer, uint32\_t length )

This is a non-blocking function, which returns directly without waiting for all data to be received. When all data is received, the module driver notifies the upper layer through a RX callback function and passes the

status parameter [kStatus\\_SerialManager\\_Success](#). This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

#### Note

The function [SerialManager\\_ReadBlocking](#) and the function [SerialManager\\_ReadNonBlocking](#) cannot be used at the same time. And, the RX callback is mandatory before the function could be used.

#### Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer.             |
| <i>buffer</i>     | Start address of the data to store the received data. |
| <i>length</i>     | The length of the data to be received.                |

#### Return values

|                                      |                                                                      |
|--------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                                                   |

### 26.5.11 **serial\_manager\_status\_t SerialManager\_TryRead ( serial\_read\_handle\_t readHandle, uint8\_t \* buffer, uint32\_t length, uint32\_t \* receivedLength )**

The function tries to read data from internal ring buffer. If the ring buffer is not empty, the data will be copied from ring buffer to up layer buffer. The copied length is the minimum of the ring buffer and up layer length. After the data is copied, the actual data length is passed by the parameter length. And There can only one buffer for receiving for the reading handle at the same time.

#### Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>readHandle</i>     | The serial manager module handle pointer.             |
| <i>buffer</i>         | Start address of the data to store the received data. |
| <i>length</i>         | The length of the data to be received.                |
| <i>receivedLength</i> | Length received from the ring buffer directly.        |

Return values

|                                            |                                                                      |
|--------------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_-<br/>Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                                                   |

### 26.5.12 serial\_manager\_status\_t SerialManager\_CancelWriting ( serial\_write\_handle\_t writeHandle )

The function cancels unfinished send transmission. When the transfer is canceled, the module notifies the upper layer through a TX callback function and passes the status parameter [kStatus\\_SerialManager\\_Canceled](#).

Note

The function [SerialManager\\_CancelWriting](#) cannot be used to abort the transmission of the function [SerialManager\\_WriteBlocking](#).

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
|--------------------|-------------------------------------------|

Return values

|                                            |                                     |
|--------------------------------------------|-------------------------------------|
| <i>kStatus_SerialManager_-<br/>Success</i> | Get successfully abort the sending. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                  |

### 26.5.13 serial\_manager\_status\_t SerialManager\_CancelReading ( serial\_read\_handle\_t readHandle )

The function cancels unfinished receive transmission. When the transfer is canceled, the module notifies the upper layer through a RX callback function and passes the status parameter [kStatus\\_SerialManager\\_Canceled](#).

## Note

The function [SerialManager\\_CancelReading](#) cannot be used to abort the transmission of the function [SerialManager\\_ReadBlocking](#).

## Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer. |
|-------------------|-------------------------------------------|

## Return values

|                                      |                                       |
|--------------------------------------|---------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Get successfully abort the receiving. |
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                    |

#### 26.5.14 **serial\_manager\_status\_t SerialManager\_InstallTxCallback ( serial\_write\_handle\_t writeHandle, serial\_manager\_callback\_t callback, void \* callbackParam )**

This function is used to install the TX callback and callback parameter for the serial manager module. When any status of TX transmission changed, the driver will notify the upper layer by the installed callback function. And the status is also passed as status parameter when the callback is called.

## Parameters

|                      |                                           |
|----------------------|-------------------------------------------|
| <i>writeHandle</i>   | The serial manager module handle pointer. |
| <i>callback</i>      | The callback function.                    |
| <i>callbackParam</i> | The parameter of the callback function.   |

## Return values

|                                      |                                    |
|--------------------------------------|------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Successfully install the callback. |
|--------------------------------------|------------------------------------|

#### 26.5.15 **serial\_manager\_status\_t SerialManager\_InstallRxCallback ( serial\_read\_handle\_t readHandle, serial\_manager\_callback\_t callback, void \* callbackParam )**

This function is used to install the RX callback and callback parameter for the serial manager module. When any status of RX transmission changed, the driver will notify the upper layer by the installed callback

function. And the status is also passed as status parameter when the callback is called.

## Parameters

|                      |                                           |
|----------------------|-------------------------------------------|
| <i>readHandle</i>    | The serial manager module handle pointer. |
| <i>callback</i>      | The callback function.                    |
| <i>callbackParam</i> | The parameter of the callback function.   |

## Return values

|                                      |                                    |
|--------------------------------------|------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Successfully install the callback. |
|--------------------------------------|------------------------------------|

**26.5.16 static bool SerialManager\_needPollingIsr ( void ) [inline], [static]**

This function is used to check if need polling ISR.

## Return values

|             |                  |
|-------------|------------------|
| <i>TRUE</i> | if need polling. |
|-------------|------------------|

**26.5.17 serial\_manager\_status\_t SerialManager\_EnterLowpower ( serial\_handle\_t serialHandle )**

This function is used to prepare to enter low power consumption.

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SerialManager_Success</i> | Successful operation. |
|--------------------------------------|-----------------------|

**26.5.18 serial\_manager\_status\_t SerialManager\_ExitLowpower ( serial\_handle\_t serialHandle )**

This function is used to restore from low power consumption.

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SerialManager_Success</i> | Successful operation. |
|--------------------------------------|-----------------------|

### 26.5.19 void SerialManager\_SetLowpowerCriticalCb ( const serial\_manager\_lowpower\_critical\_CBs\_t \* *pfCallback* )

## Parameters

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>pfCallback</i> | Pointer to the function structure used to allow/disable lowpower. |
|-------------------|-------------------------------------------------------------------|

## 26.6 Serial Port Uart

### 26.6.1 Overview

#### Macros

- #define `SERIAL_PORT_UART_DMA_RECEIVE_DATA_LENGTH` (64U)  
*serial port uart handle size*
- #define `SERIAL_USE_CONFIGURE_STRUCTURE` (0U)  
*Enable or disable the configure structure pointer.*

#### Enumerations

- enum `serial_port_uart_parity_mode_t` {  
    `kSerialManager_UartParityDisabled` = 0x0U,  
    `kSerialManager_UartParityEven` = 0x2U,  
    `kSerialManager_UartParityOdd` = 0x3U }  
*serial port uart parity mode*
- enum `serial_port_uart_stop_bit_count_t` {  
    `kSerialManager_UartOneStopBit` = 0U,  
    `kSerialManager_UartTwoStopBit` = 1U }  
*serial port uart stop bit count*

### 26.6.2 Enumeration Type Documentation

#### 26.6.2.1 enum serial\_port\_uart\_parity\_mode\_t

Enumerator

***kSerialManager\_UartParityDisabled*** Parity disabled.  
***kSerialManager\_UartParityEven*** Parity even enabled.  
***kSerialManager\_UartParityOdd*** Parity odd enabled.

#### 26.6.2.2 enum serial\_port\_uart\_stop\_bit\_count\_t

Enumerator

***kSerialManager\_UartOneStopBit*** One stop bit.  
***kSerialManager\_UartTwoStopBit*** Two stop bits.



## 26.7 Serial Port SWO

### 26.7.1 Overview

#### Data Structures

- struct [serial\\_port\\_swo\\_config\\_t](#)  
*serial port swo config struct [More...](#)*

#### Macros

- #define [SERIAL\\_PORT\\_SWO\\_HANDLE\\_SIZE](#) (12U)  
*serial port swo handle size*

#### Enumerations

- enum [serial\\_port\\_swo\\_protocol\\_t](#) {  
    [kSerialManager\\_SwoProtocolManchester](#) = 1U,  
    [kSerialManager\\_SwoProtocolNrz](#) = 2U }  
*serial port swo protocol*

### 26.7.2 Data Structure Documentation

#### 26.7.2.1 struct serial\_port\_swo\_config\_t

##### Data Fields

- uint32\_t [clockRate](#)  
*clock rate*
- uint32\_t [baudRate](#)  
*baud rate*
- uint32\_t [port](#)  
*Port used to transfer data.*
- [serial\\_port\\_swo\\_protocol\\_t](#) [protocol](#)  
*SWO protocol.*

### 26.7.3 Enumeration Type Documentation

#### 26.7.3.1 enum serial\_port\_swo\_protocol\_t

##### Enumerator

***kSerialManager\_SwoProtocolManchester*** SWO Manchester protocol.  
***kSerialManager\_SwoProtocolNrz*** SWO UART/NRZ protocol.

## Chapter 27

### Enet\_cmsis\_driver

This section describes the programming interface of the ENET Cortex Microcontroller Software Interface Standard (CMSIS) driver. This driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The ENET CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

#### 27.1 Typical use case

```
void ENET_SignalEvent_t(uint32_t event)
{
 if (event == ARM_ETH_MAC_EVENT_RX_FRAME)
 {
 uint32_t size;
 uint32_t len;

 /* Get the Frame size */
 size = EXAMPLE_ENET.GetRxFrameSize();
 /* Call ENET_ReadFrame when there is a received frame. */
 if (size != 0)
 {
 /* Received valid frame. Deliver the rx buffer with the size equal to length. */
 uint8_t *data = (uint8_t *)malloc(size);
 if (data)
 {
 len = EXAMPLE_ENET.ReadFrame(data, size);
 if (size == len)
 {
 /* Increase the received frame numbers. */
 if (g_rxIndex < ENET_EXAMPLE_LOOP_COUNT)
 {
 g_rxIndex++;
 }
 }
 free(data);
 }
 }
 }
 if (event == ARM_ETH_MAC_EVENT_TX_FRAME)
 {
 g_testTxNum ++;
 }
}

/* Initialize the ENET module. */
EXAMPLE_ENET.Initialize(ENET_SignalEvent_t);
```

```

EXAMPLE_ENET.PowerControl(ARM_POWER_FULL);
EXAMPLE_ENET.SetMacAddress((ARM_ETH_MAC_ADDR *)g_macAddr);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONFIGURE, linkInfo.speed << ARM_ETH_MAC_SPEED_Pos |
 linkInfo.duplex << ARM_ETH_MAC_DUPLEX_Pos | ARM_ETH_MAC_ADDRESS_BROADCAST);
EXAMPLE_ENET_PHY.PowerControl(ARM_POWER_FULL);
EXAMPLE_ENET_PHY.SetMode(ARM_ETH_PHY_AUTO_NEGOTIATE);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_RX, 1);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_TX, 1);
if (EXAMPLE_ENET_PHY.GetLinkState() == ARM_ETH_LINK_UP)
{
 linkInfo = EXAMPLE_ENET_PHY.GetLinkInfo();
}
else
{
 PRINTF("\r\nPHY Link down, please check the cable connection and link partner setting.\r\n");
}

/* Build broadcast for sending. */
ENET_BuildBroadCastFrame();

while (1)
{
 /* Check the total number of received number. */
 if (g_rxCheckIdx != g_rxIndex)
 {
 PRINTF("The %d frame has been successfully received!\r\n", g_rxIndex);
 g_rxCheckIdx = g_rxIndex;
 }
 if (g_testTxNum && (g_txCheckIdx != g_testTxNum))
 {
 g_txCheckIdx = g_testTxNum;
 PRINTF("The %d frame transmitted success!\r\n", g_txCheckIdx);
 }
 /* Get the Frame size */
 if (txnumber < ENET_EXAMPLE_LOOP_COUNT)
 {
 txnumber++;
 /* Send a multicast frame when the PHY is link up. */
 if (EXAMPLE_ENET.SendFrame(&g_frame[0], ENET_DATA_LENGTH, ARM_ETH_MAC_TX_FRAME_EVENT) ==
 ARM_DRIVER_OK)
 {
 for (uint32_t count = 0; count < 0x3FF; count++)
 {
 __ASM("nop");
 }
 }
 else
 {
 PRINTF("\r\nTransmit frame failed!\r\n");
 }
 }
}

```

## 27.1.1 CODEC Adapter

### 27.1.1.1 Overview

#### Enumerations

- enum {  
[kCODEC\\_WM8904](#),  
[kCODEC\\_WM8960](#),  
[kCODEC\\_WM8524](#),  
[kCODEC\\_SGTL5000](#),  
[kCODEC\\_DA7212](#),  
[kCODEC\\_CS42888](#),  
[kCODEC\\_CS42448](#),  
[kCODEC\\_AK4497](#),  
[kCODEC\\_AK4458](#),  
[kCODEC\\_TFA9XXX](#),  
[kCODEC\\_TFA9896](#),  
[kCODEC\\_WM8962](#) }  
*codec type*

### 27.1.1.2 Enumeration Type Documentation

#### 27.1.1.2.1 anonymous enum

Enumerator

***kCODEC\_WM8904*** wm8904  
***kCODEC\_WM8960*** wm8960  
***kCODEC\_WM8524*** wm8524  
***kCODEC\_SGTL5000*** sgtl5000  
***kCODEC\_DA7212*** da7212  
***kCODEC\_CS42888*** CS42888.  
***kCODEC\_CS42448*** CS42448.  
***kCODEC\_AK4497*** AK4497.  
***kCODEC\_AK4458*** ak4458  
***kCODEC\_TFA9XXX*** tfa9xxx  
***kCODEC\_TFA9896*** tfa9896  
***kCODEC\_WM8962*** wm8962

**How to Reach Us:****Home Page:**

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

**Web Support:**

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

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

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

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.

