

Document Number: MCUXSDKAPIRM
Rev 2.15.000
Jan 2024

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	25
4.2.1	struct _ccm_analog_frac_pll_config	25
4.2.2	struct _ccm_analog_integer_pll_config	25
4.3	Macro Definition Documentation	26
4.3.1	FSL_CLOCK_DRIVER_VERSION	26
4.3.2	ECSPI_CLOCKS	26
4.3.3	ENET_CLOCKS	26
4.3.4	GPIO_CLOCKS	26
4.3.5	GPT_CLOCKS	26
4.3.6	I2C_CLOCKS	27
4.3.7	IOMUX_CLOCKS	27
4.3.8	IPMUX_CLOCKS	27
4.3.9	PWM_CLOCKS	27
4.3.10	RDC_CLOCKS	27
4.3.11	SAI_CLOCKS	28
4.3.12	RDC_SEMA42_CLOCKS	28
4.3.13	UART_CLOCKS	28
4.3.14	USDHC_CLOCKS	28
4.3.15	WDOG_CLOCKS	28
4.3.16	TMU_CLOCKS	29
4.3.17	SDMA_CLOCKS	29
4.3.18	MU_CLOCKS	29
4.3.19	QSPI_CLOCKS	29
4.3.20	PDM_CLOCKS	29
4.3.21	kCLOCK_CoreSysClk	30
4.3.22	CLOCK_GetCoreSysClkFreq	30

Section No.	Title	Page No.
4.4	Typedef Documentation	31
4.4.1	<code>clock_name_t</code>	31
4.4.2	<code>clock_ip_name_t</code>	31
4.4.3	<code>clock_root_control_t</code>	31
4.4.4	<code>clock_root_t</code>	31
4.4.5	<code>clock_rootmux_m4_clk_sel_t</code>	31
4.4.6	<code>clock_rootmux_axi_clk_sel_t</code>	31
4.4.7	<code>clock_rootmux_ahb_clk_sel_t</code>	31
4.4.8	<code>clock_rootmux_audio_ahb_clk_sel_t</code>	31
4.4.9	<code>clock_rootmux_qspi_clk_sel_t</code>	31
4.4.10	<code>clock_rootmux_ecspi_clk_sel_t</code>	31
4.4.11	<code>clock_rootmux_enet_axi_clk_sel_t</code>	31
4.4.12	<code>clock_rootmux_enet_ref_clk_sel_t</code>	31
4.4.13	<code>clock_rootmux_enet_timer_clk_sel_t</code>	31
4.4.14	<code>clock_rootmux_enet_phy_clk_sel_t</code>	31
4.4.15	<code>clock_rootmux_i2c_clk_sel_t</code>	31
4.4.16	<code>clock_rootmux_uart_clk_sel_t</code>	31
4.4.17	<code>clock_rootmux_gpt_t</code>	31
4.4.18	<code>clock_rootmux_wdog_clk_sel_t</code>	31
4.4.19	<code>clock_rootmux_Pwm_clk_sel_t</code>	31
4.4.20	<code>clock_rootmux_sai_clk_sel_t</code>	31
4.4.21	<code>clock_rootmux_pdm_clk_sel_t</code>	31
4.4.22	<code>clock_rootmux_noc_clk_sel_t</code>	31
4.4.23	<code>clock_pll_gate_t</code>	31
4.4.24	<code>clock_gate_value_t</code>	31
4.4.25	<code>clock_pll_bypass_ctrl_t</code>	31
4.4.26	<code>clock_pll_clke_t</code>	32
4.4.27	<code>ccm_analog_frac_pll_config_t</code>	32
4.4.28	<code>ccm_analog_integer_pll_config_t</code>	32
4.5	Enumeration Type Documentation	32
4.5.1	<code>_clock_name</code>	32
4.5.2	<code>_clock_ip_name</code>	33
4.5.3	<code>_clock_root_control</code>	35
4.5.4	<code>_clock_root</code>	36
4.5.5	<code>_clock_rootmux_m4_clk_sel</code>	37
4.5.6	<code>_clock_rootmux_axi_clk_sel</code>	37
4.5.7	<code>_clock_rootmux_ahb_clk_sel</code>	37
4.5.8	<code>_clock_rootmux_audio_ahb_clk_sel</code>	38
4.5.9	<code>_clock_rootmux_qspi_clk_sel</code>	38
4.5.10	<code>_clock_rootmux_ecspi_clk_sel</code>	38
4.5.11	<code>_clock_rootmux_enet_axi_clk_sel</code>	39
4.5.12	<code>_clock_rootmux_enet_ref_clk_sel</code>	39
4.5.13	<code>_clock_rootmux_enet_timer_clk_sel</code>	39
4.5.14	<code>_clock_rootmux_enet_phy_clk_sel</code>	40

Section No.	Title	Page No.
4.5.15	<code>_clock_rootmux_i2c_clk_sel</code>	40
4.5.16	<code>_clock_rootmux_uart_clk_sel</code>	40
4.5.17	<code>_clock_rootmux_gpt</code>	41
4.5.18	<code>_clock_rootmux_wdog_clk_sel</code>	41
4.5.19	<code>_clock_rootmux_pwm_clk_sel</code>	41
4.5.20	<code>_clock_rootmux_sai_clk_sel</code>	42
4.5.21	<code>_clock_rootmux_pdm_clk_sel</code>	42
4.5.22	<code>_clock_rootmux_noc_clk_sel</code>	42
4.5.23	<code>_clock_pll_gate</code>	43
4.5.24	<code>_clock_gate_value</code>	43
4.5.25	<code>_clock_pll_bypass_ctrl</code>	44
4.5.26	<code>_ccm_analog_pll_elke</code>	44
4.5.27	anonymous enum	45
4.6	Function Documentation	45
4.6.1	<code>CLOCK_SetRootMux</code>	45
4.6.2	<code>CLOCK_GetRootMux</code>	45
4.6.3	<code>CLOCK_EnableRoot</code>	46
4.6.4	<code>CLOCK_DisableRoot</code>	46
4.6.5	<code>CLOCK_IsRootEnabled</code>	46
4.6.6	<code>CLOCK_UpdateRoot</code>	46
4.6.7	<code>CLOCK_SetRootDivider</code>	47
4.6.8	<code>CLOCK_GetRootPreDivider</code>	47
4.6.9	<code>CLOCK_GetRootPostDivider</code>	47
4.6.10	<code>CLOCK_ControlGate</code>	48
4.6.11	<code>CLOCK_EnableClock</code>	48
4.6.12	<code>CLOCK_DisableClock</code>	48
4.6.13	<code>CLOCK_PowerUpPll</code>	48
4.6.14	<code>CLOCK_PowerDownPll</code>	49
4.6.15	<code>CLOCK_SetPllBypass</code>	49
4.6.16	<code>CLOCK_IsPllBypassed</code>	49
4.6.17	<code>CLOCK_IsPllLocked</code>	50
4.6.18	<code>CLOCK_EnableAnalogClock</code>	50
4.6.19	<code>CLOCK_DisableAnalogClock</code>	50
4.6.20	<code>CLOCK_OverridePllClke</code>	50
4.6.21	<code>CLOCK_OverridePllPd</code>	51
4.6.22	<code>CLOCK_InitArmPll</code>	51
4.6.23	<code>CLOCK_InitSysPll1</code>	51
4.6.24	<code>CLOCK_InitSysPll2</code>	52
4.6.25	<code>CLOCK_InitSysPll3</code>	52
4.6.26	<code>CLOCK_InitAudioPll1</code>	52
4.6.27	<code>CLOCK_InitAudioPll2</code>	53
4.6.28	<code>CLOCK_InitVideoPll1</code>	53
4.6.29	<code>CLOCK_InitIntegerPll</code>	53
4.6.30	<code>CLOCK_GetIntegerPllFreq</code>	54

Section No.	Title	Page No.
4.6.31	<code>CLOCK_InitFracPll</code>	55
4.6.32	<code>CLOCK_GetFracPllFreq</code>	55
4.6.33	<code>CLOCK_GetPllFreq</code>	55
4.6.34	<code>CLOCK_GetPllRefClkFreq</code>	56
4.6.35	<code>CLOCK_GetFreq</code>	56
4.6.36	<code>CLOCK_GetClockRootFreq</code>	56
4.6.37	<code>CLOCK_GetCoreM4Freq</code>	57
4.6.38	<code>CLOCK_GetAxiFreq</code>	57
4.6.39	<code>CLOCK_GetAhbFreq</code>	57
4.6.40	<code>CLOCK_GetEnetAxiFreq</code>	57

Chapter 5 IOMUXC: IOMUX Controller

5.1	Overview	58
5.2	Macro Definition Documentation	77
5.2.1	<code>FSL_IOMUXC_DRIVER_VERSION</code>	77
5.3	Function Documentation	77
5.3.1	<code>IOMUXC_SetPinMux</code>	77
5.3.2	<code>IOMUXC_SetPinConfig</code>	78

Chapter 6 Common Driver

6.1	Overview	79
6.2	Macro Definition Documentation	85
6.2.1	<code>FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ</code>	85
6.2.2	<code>MAKE_STATUS</code>	85
6.2.3	<code>MAKE_VERSION</code>	85
6.2.4	<code>FSL_COMMON_DRIVER_VERSION</code>	86
6.2.5	<code>DEBUG_CONSOLE_DEVICE_TYPE_NONE</code>	86
6.2.6	<code>DEBUG_CONSOLE_DEVICE_TYPE_UART</code>	86
6.2.7	<code>DEBUG_CONSOLE_DEVICE_TYPE_LPUART</code>	86
6.2.8	<code>DEBUG_CONSOLE_DEVICE_TYPE_LPSCI</code>	86
6.2.9	<code>DEBUG_CONSOLE_DEVICE_TYPE_USBCDC</code>	86
6.2.10	<code>DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM</code>	86
6.2.11	<code>DEBUG_CONSOLE_DEVICE_TYPE_IUART</code>	86
6.2.12	<code>DEBUG_CONSOLE_DEVICE_TYPE_VUSART</code>	86
6.2.13	<code>DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART</code>	86
6.2.14	<code>DEBUG_CONSOLE_DEVICE_TYPE_SWO</code>	86
6.2.15	<code>DEBUG_CONSOLE_DEVICE_TYPE_QSCI</code>	86
6.2.16	<code>MIN</code>	86
6.2.17	<code>MAX</code>	86
6.2.18	<code>ARRAY_SIZE</code>	86

Section No.	Title	Page No.
6.2.19	UINT16_MAX	86
6.2.20	UINT32_MAX	86
6.2.21	SUPPRESS_FALL_THROUGH_WARNING	86
6.2.22	SDK_SIZEALIGN	87
6.3	Typedef Documentation	87
6.3.1	status_t	87
6.4	Enumeration Type Documentation	87
6.4.1	_status_groups	87
6.4.2	anonymous enum	90
6.5	Function Documentation	90
6.5.1	SDK_Malloc	90
6.5.2	SDK_Free	90
6.5.3	SDK_DelayAtLeastUs	90
6.5.4	EnableIRQ	91
6.5.5	DisableIRQ	91
6.5.6	EnableIRQWithPriority	92
6.5.7	IRQ_SetPriority	92
6.5.8	IRQ_ClearPendingIRQ	93
6.5.9	DisableGlobalIRQ	93
6.5.10	EnableGlobalIRQ	94

Chapter 7 ECSPI: Enhanced Configurable Serial Peripheral Interface Driver

7.1	Overview	95
7.2	ECSPI Driver	96
7.2.1	Overview	96
7.2.2	Typical use case	96
7.2.3	Data Structure Documentation	102
7.2.4	Macro Definition Documentation	105
7.2.5	Typedef Documentation	105
7.2.6	Enumeration Type Documentation	105
7.2.7	Function Documentation	108
7.3	ECSPI FreeRTOS Driver	121
7.3.1	Overview	121
7.3.2	Macro Definition Documentation	121
7.3.3	Function Documentation	121
7.4	ECSPI SDMA Driver	124
7.4.1	Overview	124
7.4.2	Data Structure Documentation	125
7.4.3	Macro Definition Documentation	125

Section No.	Title	Page No.
7.4.4	Typedef Documentation	125
7.4.5	Function Documentation	125
7.5	ECSPI CMSIS Driver	129
7.5.1	Function groups	129
7.5.2	Typical use case	130

Chapter 8 ENET: Ethernet MAC Driver

8.1	Overview	131
8.2	Operations of Ethernet MAC Driver	131
8.2.1	MII interface Operation	131
8.2.2	MAC address filter	131
8.2.3	Other Basic control Operations	131
8.2.4	Transactional Operation	131
8.2.5	PTP IEEE 1588 Feature Operation	132
8.3	Typical use case	132
8.3.1	ENET Initialization, receive, and transmit operations	132
8.4	Data Structure Documentation	142
8.4.1	struct _enet_rx_bd_struct	142
8.4.2	struct _enet_tx_bd_struct	142
8.4.3	struct _enet_data_error_stats	142
8.4.4	struct _enet_rx_frame_error	143
8.4.5	struct _enet_transfer_stats	144
8.4.6	struct enet_frame_info	145
8.4.7	struct _enet_tx_dirty_ring	145
8.4.8	struct _enet_buffer_config	146
8.4.9	struct _enet_intcoalesce_config	147
8.4.10	struct _enet_avb_config	148
8.4.11	struct _enet_config	148
8.4.12	struct _enet_tx_bd_ring	151
8.4.13	struct _enet_rx_bd_ring	151
8.4.14	struct _enet_handle	152
8.5	Macro Definition Documentation	154
8.5.1	FSL_ENET_DRIVER_VERSION	154
8.5.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	154
8.5.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	154
8.5.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	154
8.5.5	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	154
8.5.6	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	154
8.5.7	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	154

Section No.	Title	Page No.
8.5.8	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	154
8.5.9	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	154
8.5.10	ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK	154
8.5.11	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	154
8.5.12	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	154
8.5.13	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	154
8.5.14	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	154
8.5.15	ENET_BUFFDESCRIPTOR_TX_READY_MASK	154
8.5.16	ENET_BUFFDESCRIPTOR_TX_SOFTWENER1_MASK	154
8.5.17	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	154
8.5.18	ENET_BUFFDESCRIPTOR_TX_SOFTWENER2_MASK	154
8.5.19	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	154
8.5.20	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	154
8.5.21	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	154
8.5.22	ENET_FRAME_MAX_FRAMELEN	155
8.5.23	ENET_FRAME_VLAN_TAGLEN	155
8.5.24	ENET_FRAME_CRC_LEN	155
8.5.25	ENET_FIFO_MIN_RX_FULL	155
8.5.26	ENET_RX_MIN_BUFFERSIZE	155
8.5.27	ENET_PHY_MAXADDRESS	155
8.5.28	ENET_TX_INTERRUPT	155
8.5.29	ENET_RX_INTERRUPT	155
8.5.30	ENET_TS_INTERRUPT	156
8.5.31	ENET_ERR_INTERRUPT	156
8.6	Typedef Documentation	156
8.6.1	enet_mii_mode_t	156
8.6.2	enet_mii_speed_t	156
8.6.3	enet_mii_duplex_t	156
8.6.4	enet_mii_write_t	156
8.6.5	enet_mii_read_t	156
8.6.6	enet_mii_extend_opcode	156
8.6.7	enet_special_control_flag_t	156
8.6.8	enet_interrupt_enable_t	156
8.6.9	enet_event_t	157
8.6.10	enet_idle_slope_t	157
8.6.11	enet_tx_accelerator_t	157
8.6.12	enet_rx_accelerator_t	157
8.6.13	enet_rx_bd_struct_t	157
8.6.14	enet_tx_bd_struct_t	157
8.6.15	enet_data_error_stats_t	157
8.6.16	enet_rx_frame_error_t	157
8.6.17	enet_transfer_stats_t	157
8.6.18	enet_frame_info_t	157
8.6.19	enet_tx_dirty_ring_t	157

Section No.	Title	Page No.
8.6.20	enet_rx_alloc_callback_t	157
8.6.21	enet_rx_free_callback_t	157
8.6.22	enet_buffer_config_t	157
8.6.23	enet_intcoalesce_config_t	158
8.6.24	enet_avb_config_t	158
8.6.25	enet_callback_t	158
8.6.26	enet_config_t	158
8.6.27	enet_tx_bd_ring_t	159
8.6.28	enet_rx_bd_ring_t	159
8.6.29	enet_isr_ring_t	159
8.7	Enumeration Type Documentation	159
8.7.1	anonymous enum	159
8.7.2	_enet_mii_mode	159
8.7.3	_enet_mii_speed	159
8.7.4	_enet_mii_duplex	160
8.7.5	_enet_mii_write	160
8.7.6	_enet_mii_read	160
8.7.7	_enet_mii_extend_opcode	160
8.7.8	_enet_special_control_flag	160
8.7.9	_enet_interrupt_enable	161
8.7.10	_enet_event	162
8.7.11	_enet_idle_slope	162
8.7.12	_enet_tx_accelerator	163
8.7.13	_enet_rx_accelerator	163
8.8	Function Documentation	163
8.8.1	ENET_GetInstance	163
8.8.2	ENET_GetDefaultConfig	163
8.8.3	ENET_Up	164
8.8.4	ENET_Init	165
8.8.5	ENET_Down	166
8.8.6	ENET_Deinit	166
8.8.7	ENET_Reset	166
8.8.8	ENET_SetMII	166
8.8.9	ENET_SetSMI	167
8.8.10	ENET_GetSMI	167
8.8.11	ENET_ReadSMIData	167
8.8.12	ENET_StartSMIWrite	168
8.8.13	ENET_StartSMIRead	168
8.8.14	ENET_MDIOWrite	169
8.8.15	ENET_MDIORead	169
8.8.16	ENET_StartExtC45SMIWriteReg	169
8.8.17	ENET_StartExtC45SMIWriteData	171
8.8.18	ENET_StartExtC45SMIReadData	171

Section No.	Title	Page No.
8.8.19	ENET_MDIOC45Write	171
8.8.20	ENET_MDIOC45Read	172
8.8.21	ENET_SetMacAddr	172
8.8.22	ENET_GetMacAddr	173
8.8.23	ENET_AddMulticastGroup	174
8.8.24	ENET_LeaveMulticastGroup	174
8.8.25	ENET_ActiveRead	174
8.8.26	ENET_EnableSleepMode	175
8.8.27	ENET_GetAccelFunction	176
8.8.28	ENET_EnableInterrupts	176
8.8.29	ENET_DisableInterrupts	176
8.8.30	ENET_GetInterruptStatus	177
8.8.31	ENET_ClearInterruptStatus	177
8.8.32	ENET_SetRxISRHandler	177
8.8.33	ENET_SetTxISRHandler	178
8.8.34	ENET_SetErrISRHandler	178
8.8.35	ENET_GetRxErrBeforeReadFrame	178
8.8.36	ENET_GetStatistics	179
8.8.37	ENET_GetRxFrameSize	179
8.8.38	ENET_ReadFrame	180
8.8.39	ENET_SendFrame	181
8.8.40	ENET_SetTxReclaim	182
8.8.41	ENET_ReclaimTxDescriptor	183
8.8.42	ENET_GetRxFrame	184
8.8.43	ENET_StartTxFrame	185
8.8.44	ENET_TransmitIRQHandler	185
8.8.45	ENET_ReceiveIRQHandler	185
8.8.46	ENET_CommonFrame1IRQHandler	186
8.8.47	ENET_CommonFrame2IRQHandler	186
8.8.48	ENET_ErrorIRQHandler	186
8.8.49	ENET_Ptp1588IRQHandler	186
8.8.50	ENET_CommonFrame0IRQHandler	187
8.9	Variable Documentation	187
8.9.1	s_enetClock	187

Chapter 9 GPC: General Power Controller Driver

9.1	Overview	188
9.2	Macro Definition Documentation	190
9.2.1	FSL_GPC_DRIVER_VERSION	190
9.3	Enumeration Type Documentation	190
9.3.1	_gpc_lpm_mode	190

Section No.	Title	Page No.
9.3.2	<code>_gpc_pgc_ack_sel</code>	190
9.3.3	<code>_gpc_standby_count</code>	190
9.4	Function Documentation	190
9.4.1	<code>GPC_AllowIRQs</code>	190
9.4.2	<code>GPC_DisallowIRQs</code>	191
9.4.3	<code>GPC_GetLpmMode</code>	191
9.4.4	<code>GPC_EnableIRQ</code>	191
9.4.5	<code>GPC_DisableIRQ</code>	191
9.4.6	<code>GPC_GetIRQStatusFlag</code>	192
9.4.7	<code>GPC_DsmTriggerMask</code>	192
9.4.8	<code>GPC_WFIMask</code>	192
9.4.9	<code>GPC_SelectPGCAckSignal</code>	192
9.4.10	<code>GPC_PowerDownRequestMask</code>	193
9.4.11	<code>GPC_PGCMapping</code>	193
9.4.12	<code>GPC_TimeSlotConfigureForPUS</code>	193
9.4.13	<code>GPC_EnterWaitMode</code>	193
9.4.14	<code>GPC_EnterStopMode</code>	194
9.4.15	<code>GPC_Init</code>	194

Chapter 10 GPT: General Purpose Timer

10.1	Overview	195
10.2	Function groups	195
10.2.1	<code>Initialization and deinitialization</code>	195
10.3	Typical use case	195
10.3.1	<code>GPT interrupt example</code>	195
10.4	Data Structure Documentation	199
10.4.1	<code>struct _gpt_init_config</code>	199
10.5	Typedef Documentation	199
10.5.1	<code>gpt_clock_source_t</code>	199
10.5.2	<code>gpt_input_capture_channel_t</code>	200
10.5.3	<code>gpt_input_operation_mode_t</code>	200
10.5.4	<code>gpt_output_compare_channel_t</code>	200
10.5.5	<code>gpt_output_operation_mode_t</code>	200
10.5.6	<code>gpt_status_flag_t</code>	200
10.5.7	<code>gpt_config_t</code>	200
10.6	Enumeration Type Documentation	200
10.6.1	<code>_gpt_clock_source</code>	200
10.6.2	<code>_gpt_input_capture_channel</code>	200
10.6.3	<code>_gpt_input_operation_mode</code>	201

Section No.	Title	Page No.
10.6.4	<code>_gpt_output_compare_channel</code>	201
10.6.5	<code>_gpt_output_operation_mode</code>	201
10.6.6	<code>_gpt_interrupt_enable</code>	201
10.6.7	<code>_gpt_status_flag</code>	202
10.7	Function Documentation	202
10.7.1	<code>GPT_Init</code>	202
10.7.2	<code>GPT_Deinit</code>	202
10.7.3	<code>GPT_GetDefaultConfig</code>	202
10.7.4	<code>GPT_SoftwareReset</code>	203
10.7.5	<code>GPT_SetClockSource</code>	203
10.7.6	<code>GPT_GetClockSource</code>	203
10.7.7	<code>GPT_SetClockDivider</code>	203
10.7.8	<code>GPT_GetClockDivider</code>	204
10.7.9	<code>GPT_SetOscClockDivider</code>	204
10.7.10	<code>GPT_GetOscClockDivider</code>	204
10.7.11	<code>GPT_StartTimer</code>	204
10.7.12	<code>GPT_StopTimer</code>	205
10.7.13	<code>GPT_GetCurrentTimerCount</code>	205
10.7.14	<code>GPT_SetInputOperationMode</code>	205
10.7.15	<code>GPT_GetInputOperationMode</code>	205
10.7.16	<code>GPT_GetInputCaptureValue</code>	206
10.7.17	<code>GPT_SetOutputOperationMode</code>	206
10.7.18	<code>GPT_GetOutputOperationMode</code>	207
10.7.19	<code>GPT_SetOutputCompareValue</code>	207
10.7.20	<code>GPT_GetOutputCompareValue</code>	207
10.7.21	<code>GPT_ForceOutput</code>	208
10.7.22	<code>GPT_EnableInterrupts</code>	208
10.7.23	<code>GPT_DisableInterrupts</code>	208
10.7.24	<code>GPT_GetEnabledInterrupts</code>	208
10.7.25	<code>GPT_GetStatusFlags</code>	209
10.7.26	<code>GPT_ClearStatusFlags</code>	209

Chapter 11 GPIO: General-Purpose Input/Output Driver

11.1	Overview	210
11.2	Typical use case	210
11.2.1	<code>Input Operation</code>	210
11.3	Data Structure Documentation	212
11.3.1	<code>struct _gpio_pin_config</code>	212
11.4	Macro Definition Documentation	212
11.4.1	<code>FSL_GPIO_DRIVER_VERSION</code>	212

Section No.	Title	Page No.
11.5	Typedef Documentation	212
11.5.1	gpio_pin_direction_t	212
11.5.2	gpio_interrupt_mode_t	212
11.5.3	gpio_pin_config_t	212
11.6	Enumeration Type Documentation	213
11.6.1	_gpio_pin_direction	213
11.6.2	_gpio_interrupt_mode	213
11.7	Function Documentation	213
11.7.1	GPIO_PinInit	213
11.7.2	GPIO_PinWrite	213
11.7.3	GPIO_WritePinOutput	214
11.7.4	GPIO_PortSet	214
11.7.5	GPIO_SetPinsOutput	214
11.7.6	GPIO_PortClear	214
11.7.7	GPIO_ClearPinsOutput	215
11.7.8	GPIO_PortToggle	215
11.7.9	GPIO_PinRead	215
11.7.10	GPIO_ReadPinInput	215
11.7.11	GPIO_PinReadPadStatus	216
11.7.12	GPIO_ReadPadStatus	217
11.7.13	GPIO_PinSetInterruptConfig	217
11.7.14	GPIO_SetPinInterruptConfig	217
11.7.15	GPIO_PortEnableInterrupts	217
11.7.16	GPIO_EnableInterrupts	218
11.7.17	GPIO_PortDisableInterrupts	218
11.7.18	GPIO_DisableInterrupts	218
11.7.19	GPIO_PortGetInterruptFlags	218
11.7.20	GPIO_GetPinsInterruptFlags	219
11.7.21	GPIO_PortClearInterruptFlags	219
11.7.22	GPIO_ClearPinsInterruptFlags	219

Chapter 12 I2C: Inter-Integrated Circuit Driver

12.1	Overview	221
12.2	I2C Driver	222
12.2.1	Overview	222
12.2.2	Typical use case	222
12.2.3	Data Structure Documentation	226
12.2.4	Macro Definition Documentation	230
12.2.5	Typedef Documentation	230
12.2.6	Enumeration Type Documentation	231
12.2.7	Function Documentation	233

Section No.	Title	Page No.
12.3 I2C FreeRTOS Driver		245
12.3.1 Overview		245
12.3.2 Macro Definition Documentation		245
12.3.3 Function Documentation		245
12.4 I2C CMSIS Driver		248
12.4.1 I2C CMSIS Driver		248
 Chapter 13 PWM: Pulse Width Modulation Driver		
13.1 Overview		250
13.2 PWM Driver		250
13.2.1 Initialization and deinitialization		250
13.3 Typical use case		250
13.3.1 PWM output		250
13.4 Typedef Documentation		253
13.4.1 <code>pwm_clock_source_t</code>		253
13.4.2 <code>pwm_fifo_water_mark_t</code>		253
13.4.3 <code>pwm_byte_data_swap_t</code>		253
13.4.4 <code>pwm_half_word_data_swap_t</code>		253
13.5 Enumeration Type Documentation		253
13.5.1 <code>_pwm_clock_source</code>		253
13.5.2 <code>_pwm_fifo_water_mark</code>		253
13.5.3 <code>_pwm_byte_data_swap</code>		254
13.5.4 <code>_pwm_half_word_data_swap</code>		254
13.5.5 <code>_pwm_output_configuration</code>		254
13.5.6 <code>_pwm_sample_repeat</code>		254
13.5.7 <code>_pwm_interrupt_enable</code>		255
13.5.8 <code>_pwm_status_flags</code>		255
13.5.9 <code>_pwm_fifo_available</code>		255
13.6 Function Documentation		255
13.6.1 <code>PWM_Init</code>		255
13.6.2 <code>PWM_Deinit</code>		256
13.6.3 <code>PWM_GetDefaultConfig</code>		256
13.6.4 <code>PWM_StartTimer</code>		256
13.6.5 <code>PWM_StopTimer</code>		257
13.6.6 <code>PWM_SoftwareReset</code>		257
13.6.7 <code>PWM_EnableInterrupts</code>		257
13.6.8 <code>PWM_DisableInterrupts</code>		257
13.6.9 <code>PWM_GetEnabledInterrupts</code>		258
13.6.10 <code>PWM_GetStatusFlags</code>		258

Section No.	Title	Page No.
13.6.11	PWM_clearStatusFlags	258
13.6.12	PWM_GetFIFOAvailable	259
13.6.13	PWM_SetSampleValue	259
13.6.14	PWM_GetSampleValue	259
13.6.15	PWM_SetPeriodValue	259
13.6.16	PWM_GetPeriodValue	260
13.6.17	PWM_GetCounterValue	260

Chapter 14 UART: Universal Asynchronous Receiver/Transmitter Driver

14.1	Overview	261
14.2	UART Driver	262
14.2.1	Overview	262
14.2.2	Typical use case	262
14.2.3	Data Structure Documentation	268
14.2.4	Macro Definition Documentation	272
14.2.5	Typedef Documentation	272
14.2.6	Enumeration Type Documentation	272
14.2.7	Function Documentation	275
14.2.8	Variable Documentation	288
14.3	UART FreeRTOS Driver	289
14.3.1	Overview	289
14.3.2	Data Structure Documentation	289
14.3.3	Macro Definition Documentation	290
14.3.4	Function Documentation	290
14.4	UART SDMA Driver	292
14.4.1	Overview	292
14.4.2	Data Structure Documentation	293
14.4.3	Macro Definition Documentation	294
14.4.4	Typedef Documentation	294
14.4.5	Function Documentation	294
14.5	UART CMSIS Driver	298
14.5.1	Function groups	298

Chapter 15 MU: Messaging Unit

15.1	Overview	300
15.2	Function description	300
15.2.1	MU initialization	300
15.2.2	MU message	300

Section No.	Title	Page No.
15.2.3	MU flags	301
15.2.4	Status and interrupt	301
15.2.5	MU misc functions	301
15.3	Macro Definition Documentation	304
15.3.1	FSL_MU_DRIVER_VERSION	304
15.4	Enumeration Type Documentation	304
15.4.1	_mu_status_flags	304
15.4.2	_mu_interrupt_enable	304
15.4.3	_mu_interrupt_trigger	305
15.5	Function Documentation	305
15.5.1	MU_Init	305
15.5.2	MU_Deinit	305
15.5.3	MU_SendMsgNonBlocking	305
15.5.4	MU_SendMsg	306
15.5.5	MU_ReceiveMsgNonBlocking	306
15.5.6	MU_ReceiveMsg	307
15.5.7	MU_SetFlagsNonBlocking	308
15.5.8	MU_SetFlags	308
15.5.9	MU_GetFlags	309
15.5.10	MU_GetStatusFlags	309
15.5.11	MU_GetInterruptsPending	310
15.5.12	MU_ClearStatusFlags	311
15.5.13	MU_EnableInterrupts	312
15.5.14	MU_DisableInterrupts	312
15.5.15	MU_TriggerInterrupts	312
15.5.16	MU_MaskHardwareReset	313
15.5.17	MU_GetOtherCorePowerMode	313

Chapter 16 PDM: Microphone Interface

16.1	Overview	314
16.2	Typical use case	314
16.3	PDM Driver	315
16.3.1	Overview	315
16.3.2	Typical use case	315
16.3.3	Data Structure Documentation	324
16.3.4	Enumeration Type Documentation	326
16.3.5	Function Documentation	331
16.4	PDM SDMA Driver	348
16.4.1	Typical use case	348

Section No.	Title	Page No.
16.4.2	Overview	348
16.4.3	Data Structure Documentation	349
16.4.4	Function Documentation	350

Chapter 17 RDC: Resource Domain Controller

17.1	Overview	353
17.2	Data Structure Documentation	355
17.2.1	struct _rdc_hardware_config	355
17.2.2	struct _rdc_domain_assignment	355
17.2.3	struct _rdc_periph_access_config	356
17.2.4	struct _rdc_mem_access_config	356
17.2.5	struct _rdc_mem_status	357
17.3	Typedef Documentation	357
17.3.1	rdc_mem_access_config_t	357
17.4	Enumeration Type Documentation	357
17.4.1	_rdc_interrupts	357
17.4.2	_rdc_flags	358
17.4.3	_rdc_access_policy	358
17.5	Function Documentation	358
17.5.1	RDC_Init	358
17.5.2	RDC_Deinit	358
17.5.3	RDC_GetHardwareConfig	358
17.5.4	RDC_EnableInterrupts	359
17.5.5	RDC_DisableInterrupts	359
17.5.6	RDC_GetInterruptStatus	359
17.5.7	RDC_ClearInterruptStatus	359
17.5.8	RDC_GetStatus	360
17.5.9	RDC_ClearStatus	360
17.5.10	RDC_SetMasterDomainAssignment	360
17.5.11	RDC_GetDefaultMasterDomainAssignment	361
17.5.12	RDC_LockMasterDomainAssignment	361
17.5.13	RDC_SetPeriphAccessConfig	361
17.5.14	RDC_GetDefaultPeriphAccessConfig	361
17.5.15	RDC_LockPeriphAccessConfig	362
17.5.16	RDC_GetPeriphAccessPolicy	362
17.5.17	RDC_SetMemAccessConfig	362
17.5.18	RDC_GetDefaultMemAccessConfig	363
17.5.19	RDC_LockMemAccessConfig	363
17.5.20	RDC_SetMemAccessValid	363
17.5.21	RDC_GetMemViolationStatus	364

Section No.	Title	Page No.
17.5.22	RDC_ClearMemViolationFlag	364
17.5.23	RDC_GetMemAccessPolicy	364
17.5.24	RDC_GetCurrentMasterDomainId	364

Chapter 18 RDC_SEMA42: Hardware Semaphores Driver

18.1	Overview	366
18.2	Macro Definition Documentation	367
18.2.1	RDC_SEMA42_GATE_NUM_RESET_ALL	367
18.2.2	RDC_SEMA42_GATEEn	367
18.2.3	RDC_SEMA42_GATE_COUNT	367
18.3	Function Documentation	367
18.3.1	RDC_SEMA42_Init	367
18.3.2	RDC_SEMA42_Deinit	367
18.3.3	RDC_SEMA42_TryLock	368
18.3.4	RDC_SEMA42_Lock	368
18.3.5	RDC_SEMA42_Unlock	369
18.3.6	RDC_SEMA42_GetLockMasterIndex	369
18.3.7	RDC_SEMA42_GetLockDomainID	369
18.3.8	RDC_SEMA42_ResetGate	370
18.3.9	RDC_SEMA42_ResetAllGates	371

Chapter 19 SAI: Serial Audio Interface

19.1	Overview	372
19.2	Typical configurations	372
19.3	Typical use case	373
19.3.1	SAI Send/receive using an interrupt method	373
19.3.2	SAI Send/receive using a DMA method	373
19.4	Typical use case	373
19.5	SAI Driver	374
19.5.1	Overview	374
19.5.2	Data Structure Documentation	382
19.5.3	Macro Definition Documentation	387
19.5.4	Enumeration Type Documentation	387
19.5.5	Function Documentation	391
19.6	SAI SDMA Driver	417
19.6.1	Typical use case	417
19.6.2	Overview	417

Section No.	Title	Page No.
19.6.3	Data Structure Documentation	418
19.6.4	Function Documentation	419
Chapter 20 SDMA: Smart Direct Memory Access (SDMA) Controller Driver		
20.1	Overview	423
20.2	Typical use case	423
20.2.1	SDMA Operation	423
20.3	Data Structure Documentation	430
20.3.1	struct _sdma_config	430
20.3.2	struct _sdma_multi_fifo_config	430
20.3.3	struct _sdma_sw_done_config	430
20.3.4	struct _sdma_p2p_config	431
20.3.5	struct _sdma_transfer_config	431
20.3.6	struct _sdma_buffer_descriptor	432
20.3.7	struct _sdma_channel_control	433
20.3.8	struct _sdma_context_data	433
20.3.9	struct _sdma_handle	433
20.4	Macro Definition Documentation	434
20.4.1	FSL_SDMA_DRIVER_VERSION	434
20.5	Typedef Documentation	434
20.5.1	sdma_clock_ratio_t	434
20.5.2	sdma_config_t	434
20.5.3	sdma_multi_fifo_config_t	434
20.5.4	sdma_sw_done_config_t	434
20.5.5	sdma_transfer_config_t	434
20.5.6	sdma_buffer_descriptor_t	435
20.5.7	sdma_context_data_t	435
20.5.8	sdma_callback	435
20.6	Enumeration Type Documentation	435
20.6.1	_sdma_transfer_size	435
20.6.2	_sdma_bd_status	435
20.6.3	_sdma_bd_command	435
20.6.4	_sdma_context_switch_mode	436
20.6.5	_sdma_clock_ratio	436
20.6.6	_sdma_transfer_type	436
20.6.7	sdma_peripheral	436
20.6.8	anonymous enum	437
20.6.9	anonymous enum	437
20.6.10	anonymous enum	437

Section No.	Title	Page No.
20.6.11	anonymous enum	438
20.6.12	_sdma_done_src	438
20.7	Function Documentation	439
20.7.1	SDMA_Init	439
20.7.2	SDMA_Deinit	439
20.7.3	SDMA_GetDefaultConfig	439
20.7.4	SDMA_ResetModule	440
20.7.5	SDMA_EnableChannelErrorInterrupts	440
20.7.6	SDMA_DisableChannelErrorInterrupts	440
20.7.7	SDMA_ConfigBufferDescriptor	440
20.7.8	SDMA_SetChannelPriority	441
20.7.9	SDMA_SetSourceChannel	441
20.7.10	SDMA_StartChannelSoftware	442
20.7.11	SDMA_StartChannelEvents	442
20.7.12	SDMA_StopChannel	442
20.7.13	SDMA_SetContextSwitchMode	443
20.7.14	SDMA_GetChannelInterruptStatus	443
20.7.15	SDMA_ClearChannelInterruptStatus	443
20.7.16	SDMA_GetChannelStopStatus	443
20.7.17	SDMA_ClearChannelStopStatus	444
20.7.18	SDMA_GetChannelPendStatus	444
20.7.19	SDMA_ClearChannelPendStatus	444
20.7.20	SDMA_GetErrorStatus	445
20.7.21	SDMA_GetRequestSourceStatus	445
20.7.22	SDMA_CreateHandle	445
20.7.23	SDMA_InstallBDMemory	446
20.7.24	SDMA_SetCallback	446
20.7.25	SDMA_SetMultiFifoConfig	446
20.7.26	SDMA_EnableSwDone	447
20.7.27	SDMA_SetDoneConfig	447
20.7.28	SDMA_LoadScript	447
20.7.29	SDMA_DumpScript	448
20.7.30	SDMA_GetRamScriptVersion	448
20.7.31	SDMA_PrepTransfer	448
20.7.32	SDMA_PrepP2PTransfer	449
20.7.33	SDMA_SubmitTransfer	450
20.7.34	SDMA_StartTransfer	450
20.7.35	SDMA_StopTransfer	450
20.7.36	SDMA_AbortTransfer	451
20.7.37	SDMA_GetTransferredBytes	451
20.7.38	SDMA_IsPeripheralInSPBA	451
20.7.39	SDMA_HandleIRQ	452

Section No.	Title	Page No.
Chapter 21 SEMA4: Hardware Semaphores Driver		
21.1 Overview		453
21.2 Macro Definition Documentation		454
21.2.1 SEMA4_GATE_NUM_RESET_ALL		454
21.3 Function Documentation		454
21.3.1 SEMA4_Init		454
21.3.2 SEMA4_Deinit		454
21.3.3 SEMA4_TryLock		454
21.3.4 SEMA4_Lock		455
21.3.5 SEMA4_Unlock		455
21.3.6 SEMA4_GetLockProc		455
21.3.7 SEMA4_ResetGate		456
21.3.8 SEMA4_ResetAllGates		456
21.3.9 SEMA4_EnableGateNotifyInterrupt		457
21.3.10 SEMA4_DisableGateNotifyInterrupt		457
21.3.11 SEMA4_GetGateNotifyStatus		457
21.3.12 SEMA4_ResetGateNotify		458
21.3.13 SEMA4_ResetAllGateNotify		458
Chapter 22 TMU: Thermal Management Unit Driver		
22.1 Overview		460
22.2 Typical use case		460
22.2.1 Monitor and report Configuration		460
22.3 Data Structure Documentation		463
22.3.1 struct _tmu_thresold_config		463
22.3.2 struct _tmu_interrupt_status		464
22.3.3 struct _tmu_config		464
22.4 Macro Definition Documentation		464
22.4.1 FSL_TMU_DRIVER_VERSION		464
22.5 Enumeration Type Documentation		464
22.5.1 _tmu_interrupt_enable		464
22.5.2 _tmu_interrupt_status_flags		465
22.5.3 _tmu_average_low_pass_filter		465
22.5.4 _tmu_amplifier_gain		465
22.5.5 _tmu_amplifier_reference_voltage		466
22.6 Function Documentation		466
22.6.1 TMU_Init		466

Section No.	Title	Page No.
22.6.2	TMU_Deinit	467
22.6.3	TMU_GetDefaultConfig	467
22.6.4	TMU_Enable	467
22.6.5	TMU_EnableInterrupts	467
22.6.6	TMU_DisableInterrupts	468
22.6.7	TMU_GetInterruptStatusFlags	468
22.6.8	TMU_ClearInterruptStatusFlags	468
22.6.9	TMU_GetImmediateTemperature	468
22.6.10	TMU_GetAverageTemperature	469
22.6.11	TMU_SetHighTemperatureThresold	469

Chapter 23 WDOG: Watchdog Timer Driver

23.1	Overview	471
23.2	Typical use case	471
23.3	Data Structure Documentation	472
23.3.1	struct _wdog_work_mode	472
23.3.2	struct _wdog_config	472
23.4	Typedef Documentation	473
23.4.1	wdog_work_mode_t	473
23.4.2	wdog_config_t	473
23.5	Enumeration Type Documentation	473
23.5.1	_wdog_interrupt_enable	473
23.5.2	_wdog_status_flags	473
23.6	Function Documentation	474
23.6.1	WDOG_GetDefaultConfig	474
23.6.2	WDOG_Init	474
23.6.3	WDOG_Deinit	475
23.6.4	WDOG_Enable	475
23.6.5	WDOG_Disable	475
23.6.6	WDOG_TriggerSystemSoftwareReset	475
23.6.7	WDOG_TriggerSoftwareSignal	476
23.6.8	WDOG_EnableInterrupts	476
23.6.9	WDOG_GetStatusFlags	476
23.6.10	WDOG_ClearInterruptStatus	477
23.6.11	WDOG_SetTimeoutValue	477
23.6.12	WDOG_SetInterruptTimeoutValue	478
23.6.13	WDOG_DisablePowerDownEnable	478
23.6.14	WDOG_Refresh	478

Section No.	Title	Page No.
Chapter 24 Debug Console		
24.1 Overview	480	
24.2 Function groups	480	
24.2.1 Initialization	480	
24.2.2 Advanced Feature	481	
24.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART	485	
24.3 Typical use case	486	
24.4 Macro Definition Documentation	488	
24.4.1 DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	488	
24.4.2 DEBUGCONSOLE_REDIRECT_TO_SDK	488	
24.4.3 DEBUGCONSOLE_DISABLE	488	
24.4.4 SDK_DEBUGCONSOLE	488	
24.4.5 PRINTF	488	
24.5 Function Documentation	488	
24.5.1 DbgConsole_Init	488	
24.5.2 DbgConsole_Deinit	489	
24.5.3 DbgConsole_EnterLowpower	489	
24.5.4 DbgConsole_ExitLowpower	490	
24.5.5 DbgConsole_Printf	490	
24.5.6 DbgConsole_Vprintf	490	
24.5.7 DbgConsole_Putchar	490	
24.5.8 DbgConsole_Scanf	491	
24.5.9 DbgConsole_Getchar	491	
24.5.10 DbgConsole_BlockingPrintf	492	
24.5.11 DbgConsole_BlockingVprintf	492	
24.5.12 DbgConsole_Flush	492	
24.5.13 DbgConsole_TryGetchar	493	
24.6 debug console configuration	495	
24.6.1 Overview	495	
24.6.2 Macro Definition Documentation	496	
24.7 Semihosting	498	
24.7.1 Guide Semihosting for IAR	498	
24.7.2 Guide Semihosting for Keil µVision	498	
24.7.3 Guide Semihosting for MCUXpresso IDE	499	
24.7.4 Guide Semihosting for ARMGCC	499	
24.8 SWO	502	
24.8.1 Guide SWO for SDK	502	
24.8.2 Guide SWO for Keil µVision	503	

Section No.	Title	Page No.
24.8.3	Guide SWO for MCUXpresso IDE	504
24.8.4	Guide SWO for ARMGCC	504
Chapter 25 CODEC Driver		
25.1	Overview	505
25.2	CODEC Common Driver	506
25.2.1	Overview	506
25.2.2	Data Structure Documentation	511
25.2.3	Macro Definition Documentation	512
25.2.4	Typedef Documentation	512
25.2.5	Enumeration Type Documentation	512
25.2.6	Function Documentation	517
25.3	CODEC I2C Driver	523
25.3.1	Overview	523
25.3.2	Data Structure Documentation	524
25.3.3	Typedef Documentation	524
25.3.4	Enumeration Type Documentation	524
25.3.5	Function Documentation	525
25.4	AK4497 Driver	528
25.4.1	Overview	528
25.4.2	Data Structure Documentation	532
25.4.3	Macro Definition Documentation	532
25.4.4	Enumeration Type Documentation	532
25.4.5	Function Documentation	535
25.4.6	AK4497 Adapter	539
25.5	WM8524 Driver	547
25.5.1	Overview	547
25.5.2	Data Structure Documentation	548
25.5.3	Macro Definition Documentation	548
25.5.4	Typedef Documentation	548
25.5.5	Enumeration Type Documentation	548
25.5.6	Function Documentation	549
25.5.7	WM8524 Adapter	551

Chapter 26 Serial Manager

26.1	Overview	559
26.2	Data Structure Documentation	562
26.2.1	struct _serial_manager_config	562

Section No.	Title	Page No.
26.2.2	struct _serial_manager_callback_message	563
26.3	Macro Definition Documentation	563
26.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE	563
26.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE	563
26.3.3	SERIAL_MANAGER_USE_COMMON_TASK	563
26.3.4	SERIAL_MANAGER_HANDLE_SIZE	563
26.3.5	SERIAL_MANAGER_HANDLE_DEFINE	563
26.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	564
26.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE	564
26.3.8	SERIAL_MANAGER_TASK_PRIORITY	565
26.3.9	SERIAL_MANAGER_TASK_STACK_SIZE	565
26.4	Enumeration Type Documentation	565
26.4.1	_serial_port_type	565
26.4.2	_serial_manager_type	565
26.4.3	_serial_manager_status	565
26.5	Function Documentation	566
26.5.1	SerialManager_Init	566
26.5.2	SerialManager_Deinit	567
26.5.3	SerialManager_OpenWriteHandle	567
26.5.4	SerialManager_CloseWriteHandle	569
26.5.5	SerialManager_OpenReadHandle	570
26.5.6	SerialManager_CloseReadHandle	571
26.5.7	SerialManager_WriteBlocking	571
26.5.8	SerialManager_ReadBlocking	572
26.5.9	SerialManager_WriteNonBlocking	573
26.5.10	SerialManager_ReadNonBlocking	573
26.5.11	SerialManager_TryRead	574
26.5.12	SerialManager_CancelWriting	575
26.5.13	SerialManager_CancelReading	575
26.5.14	SerialManager_InstallITxCallback	576
26.5.15	SerialManager_InstallRxCallback	576
26.5.16	SerialManager_needPollingIsr	578
26.5.17	SerialManager_EnterLowpower	578
26.5.18	SerialManager_ExitLowpower	578
26.5.19	SerialManager_SetLowpowerCriticalCb	579
26.6	Serial Port Uart	580
26.6.1	Overview	580
26.6.2	Enumeration Type Documentation	580
26.7	Serial Port SWO	582
26.7.1	Overview	582

Section No.	Title	Page No.
26.7.2	Data Structure Documentation	582
26.7.3	Enumeration Type Documentation	583

Chapter 27 Enet_cmsis_driver

27.1	Typical use case	584
27.1.1	CODEC Adapter	586

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 FreeRTOSTM. 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 MCUXSDKRNN) 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, EM-BRACE, GREENCHIP, HITAG, I2C BUS,ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4M-OBILE, 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, Arm7TD-MI, 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

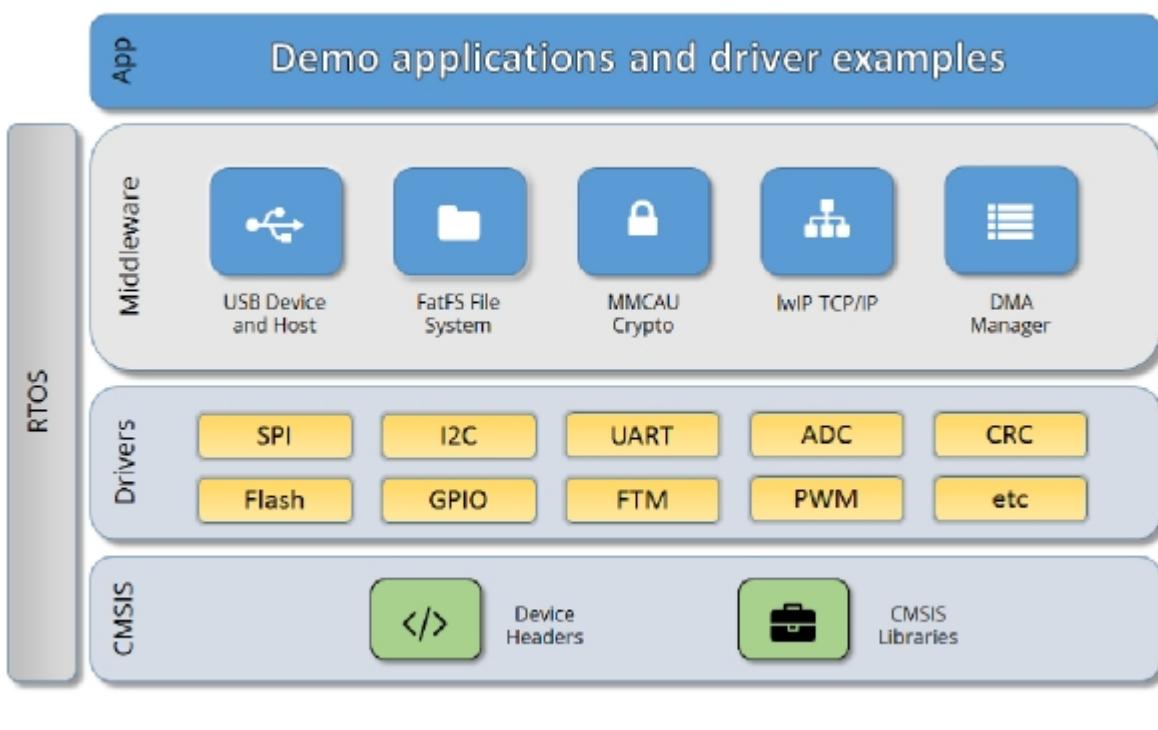
Architectural Overview

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



MCU header files

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

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler  
PUBWEAK SPI0_DriverIRQHandler  
SPI0_IRQHandler
```

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

The first level of the weak implementation are the functions defined in the vector table. In the devices/<DEVICE_NAME>/<TOOLCHAIN>/startup_<DEVICE_NAME>.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0_DriverIRQHandler) jumps to itself (B). The MCUXpresso SDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0_DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

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

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0_UART1_IRQHandler according to the use case requirements.

Feature Header Files

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

Application

See the *Getting Started with MCUXpresso SDK* document (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`
Fractional-N PLL configuration. [More...](#)
- struct `_ccm_analog_integer_pll_config`
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 TEMPSENSOR.
- #define **SDMA_CLOCKS**
Clock ip name array for SDMA.
- #define **MU_CLOCKS**
Clock ip name array for MU.
- #define **QSPI_CLOCKS**
Clock ip name array for QSPI.
- #define **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.

Typedefs

- typedef enum **_clock_name** **clock_name_t**
Clock name used to get clock frequency.
- typedef enum **_clock_ip_name** **clock_ip_name_t**
CCM CCGR gate control.
- typedef enum **_clock_root_control** **clock_root_control_t**
ccm root name used to get clock frequency.
- typedef enum **_clock_root** **clock_root_t**
ccm clock root used to get clock frequency.
- typedef enum
_clock_rootmux_m4_clk_sel **clock_rootmux_m4_clk_sel_t**
Root clock select enumeration for ARM Cortex-M4 core.
- typedef enum
_clock_rootmux_axi_clk_sel **clock_rootmux_axi_clk_sel_t**
Root clock select enumeration for AXI bus.
- typedef enum
_clock_rootmux_ahb_clk_sel **clock_rootmux_ahb_clk_sel_t**

- *Root clock select enumeration for AHB bus.*
 • **typedef enum**
`_clock_rootmux_audio_ahb_clk_sel` `clock_rootmux_audio_ahb_clk_sel_t`
Root clock select enumeration for Audio AHB bus.
- **typedef enum**
`_clock_rootmux_qspi_clk_sel` `clock_rootmux_qspi_clk_sel_t`
Root clock select enumeration for QSPI peripheral.
- **typedef enum**
`_clock_rootmux_ecspi_clk_sel` `clock_rootmux_ecspi_clk_sel_t`
Root clock select enumeration for ECSPI peripheral.
- **typedef enum**
`_clock_rootmux_enet_axi_clk_sel` `clock_rootmux_enet_axi_clk_sel_t`
Root clock select enumeration for ENET AXI bus.
- **typedef enum**
`_clock_rootmux_enet_ref_clk_sel` `clock_rootmux_enet_ref_clk_sel_t`
Root clock select enumeration for ENET REF Clcok.
- **typedef enum**
`_clock_rootmux_enet_timer_clk_sel` `clock_rootmux_enet_timer_clk_sel_t`
Root clock select enumeration for ENET TIMER Clcok.
- **typedef enum**
`_clock_rootmux_enet_phy_clk_sel` `clock_rootmux_enet_phy_clk_sel_t`
Root clock select enumeration for ENET PHY Clcok.
- **typedef enum**
`_clock_rootmux_i2c_clk_sel` `clock_rootmux_i2c_clk_sel_t`
Root clock select enumeration for I2C peripheral.
- **typedef enum**
`_clock_rootmux_uart_clk_sel` `clock_rootmux_uart_clk_sel_t`
Root clock select enumeration for UART peripheral.
- **typedef enum** `_clock_rootmux_gpt` `clock_rootmux_gpt_t`
Root clock select enumeration for GPT peripheral.
- **typedef enum**
`_clock_rootmux_wdog_clk_sel` `clock_rootmux_wdog_clk_sel_t`
Root clock select enumeration for WDOG peripheral.
- **typedef enum**
`_clock_rootmux_pwm_clk_sel` `clock_rootmux_Pwm_clk_sel_t`
Root clock select enumeration for PWM peripheral.
- **typedef enum**
`_clock_rootmux_sai_clk_sel` `clock_rootmux_sai_clk_sel_t`
Root clock select enumeration for SAI peripheral.
- **typedef enum**
`_clock_rootmux_pdm_clk_sel` `clock_rootmux_pdm_clk_sel_t`
Root clock select enumeration for PDM peripheral.
- **typedef enum**
`_clock_rootmux_noc_clk_sel` `clock_rootmux_noc_clk_sel_t`
Root clock select enumeration for NOC CLK.
- **typedef enum** `_clock_pll_gate` `clock_pll_gate_t`
CCM PLL gate control.
- **typedef enum** `_clock_gate_value` `clock_gate_value_t`
CCM gate control value.

- `typedef enum _clock_pll_bypass_ctrl clock_pll_bypass_ctrl_t`
PLL control names for PLL bypass.
- `typedef enum _ccm_analog_pll_clke clock_pll_clke_t`
PLL clock names for clock enable/disable settings.
- `typedef enum _clock_pll_ctrl clock_pll_ctrl_t`
ANALOG Power down override control.
- `typedef struct _ccm_analog_frac_pll_config ccm_analog_frac_pll_config_t`
Fractional-N PLL configuration.
- `typedef struct _ccm_analog_integer_pll_config ccm_analog_integer_pll_config_t`
Integer PLL configuration.

Enumerations

- enum `_clock_name` {

 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` { ,

```
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_Oram = CCM_TUPLE(35U, 16U),
kCLOCK_OramS = 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_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 {
    kCLOCK_RootM4,
    kCLOCK_RootAxi = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[16].TARGET_ROO-
T),
    kCLOCK_RootEnetAxi = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[17].TARGET_R-
OOT),
    kCLOCK_RootNoc = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[26].TARGET_ROO-
T),
    kCLOCK_RootAhb = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[32].TARGET_ROO-
T),
    kCLOCK_RootIpg = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[33].TARGET_ROO-
T),
    kCLOCK_RootAudioAhb,
    kCLOCK_RootAudioIpg,
    kCLOCK_RootDramAlt,
    kCLOCK_RootSai1 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[75].TARGET_ROO-
T),
    kCLOCK_RootSai2 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[76].TARGET_ROO-
T),
    kCLOCK_RootSai3 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[77].TARGET_ROO-
T),
    kCLOCK_RootSai4 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[78].TARGET_ROO-
T),
    kCLOCK_RootSai5 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[79].TARGET_ROO-
T),
    kCLOCK_RootSai6 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[80].TARGET_ROO-
T),
    kCLOCK_RootEnetRef = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[83].TARGET_R-
OOT),
    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_ROO-
T),
    kCLOCK_RootI2c1 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[90].TARGET_ROO-
T),
    kCLOCK_RootI2c2 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[91].TARGET_ROO-
T),
    kCLOCK_RootI2c3 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[92].TARGET_ROO-
T),
    kCLOCK_RootI2c4 = (uintptr_t)CCM_BASE + offsetof(CCM_Type, ROOT[93].TARGET_ROO-
T)
}

```

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_RootEcspi1,
kCLOCK_RootEcspi2,
kCLOCK_RootEcspi3,
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** {

```
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` {

```
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` {


```
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` {


```
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` {


```
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` {

```
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 {

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 {

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 {

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 Clck.

- enum _clock_rootmux_enet_timer_clk_sel {

```
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` {


```
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` {


```
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` {


```
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` {

```
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` {


```
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` {


```
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` {


```
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` {

```
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` {


```
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` {


```
kCLOCK_ArmPllGate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[12].PLL_C-
TRL),
kCLOCK_GpuPllGate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[13].PLL_C-
TRL),
kCLOCK_VpuPllGate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[14].PLL_C-
TRL),
kCLOCK_DramPllGate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[15].PLL_C-
TRL),
kCLOCK_SysPll1Gate = (uintptr_t)CCM_BASE + offsetof(CCM_Type, PLL_CTRL[16].PLL_C-
TRL),
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_C-
```

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_C-
TRL),
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** {
kCLOCK_ClockNotNeeded = 0x0U,
kCLOCK_ClockNeededRun = 0x1111U,
kCLOCK_ClockNeededRunWait = 0x2222U,
kCLOCK_ClockNeededAll = 0x3333U }

CCM gate control value.

- enum **_clock_pll_bypass_ctrl** {
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 **_ccm_analog_pll_clke** {

```

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`
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)

- static void **CLOCK_OverridePllPd** (CCM_ANALOG_Type *base, **clock_pll_ctrl_t** pdClock, bool override)
 - Override PLL clock output enable.*
- 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

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

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_Ecspi1, kCLOCK_Ecspi2,
    kCLOCK_Ecspi3, \
}
```

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:

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

4.3.7 #define IOMUX_CLOCKS

Value:

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

4.3.8 #define IPMUX_CLOCKS

Value:

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

4.3.9 #define PWM_CLOCKS

Value:

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

4.3.10 #define RDC_CLOCKS

Value:

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

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 Typedef Documentation

4.4.1 **typedef enum _clock_name clock_name_t**

4.4.2 **typedef enum _clock_ip_name clock_ip_name_t**

4.4.3 **typedef enum _clock_root_control clock_root_control_t**

4.4.4 **typedef enum _clock_root clock_root_t**

4.4.5 **typedef enum _clock_rootmux_m4_clk_sel clock_rootmux_m4_clk_sel_t**

4.4.6 **typedef enum _clock_rootmux_axi_clk_sel clock_rootmux_axi_clk_sel_t**

4.4.7 **typedef enum _clock_rootmux_ahb_clk_sel clock_rootmux_ahb_clk_sel_t**

4.4.8 **typedef enum _clock_rootmux_audio_ahb_clk_sel clock_rootmux_audio_-ahb_clk_sel_t**

4.4.9 **typedef enum _clock_rootmux_qspi_clk_sel clock_rootmux_qspi_clk_sel_t**

4.4.10 **typedef enum _clock_rootmux_ecspi_clk_sel clock_rootmux_ecspi_clk_sel_t**

4.4.11 **typedef enum _clock_rootmux_enet_axi_clk_sel clock_rootmux_enet_axi_-clk_sel_t**

4.4.12 **typedef enum _clock_rootmux_enet_ref_clk_sel clock_rootmux_enet_ref_-clk_sel_t**

4.4.13 **typedef enum _clock_rootmux_enet_timer_clk_sel clock_rootmux_enet_-timer_clk_sel_t**

4.4.14 **typedef enum _clock_rootmux_enet_phy_clk_sel clock_rootmux_enet_phy_-clk_sel_t**

4.4.15 **typedef enum _clock_rootmux_i2c_clk_sel clock_rootmux_i2c_clk_sel_t**

4.4.16 **typedef enum _clock_rootmux_uart_clk_sel clock_rootmux_uart_clk_sel_t**

4.4.17 **typedef enum _clock_rootmux_gpt clock_rootmux_gpt_t**

4.4.18 **typedef enum _clock_rootmux_wdog_clk_sel clock_rootmux_wdog_clk_sel_t**

- 0:15: REG offset to CCM_ANALOG_BASE in bytes.
- 16:20: bypass bit shift.

4.4.26 `typedef enum _ccm_analog_pll_clke 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.

4.4.27 `typedef struct _ccm_analog_frac_pll_config 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

4.4.28 `typedef struct _ccm_analog_integer_pll_config 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

4.5 Enumeration Type Documentation

4.5.1 `enum _clock_name`

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.

4.5.2 enum _clock_ip_name

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.

4.5.3 enum _clock_root_control

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 ECSP1 Clock control name.
kCLOCK_RootEcspi2 ECSP2 Clock control name.
kCLOCK_RootEcspi3 ECSP3 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.

4.5.4 enum _clock_root

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.

4.5.5 enum _clock_rootmux_m4_clk_sel

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.

4.5.6 enum _clock_rootmux_axi_clk_sel

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.

4.5.7 enum _clock_rootmux_ahb_clk_sel

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.

4.5.8 enum _clock_rootmux_audio_ahb_clk_sel

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.

4.5.9 enum _clock_rootmux_qspi_clk_sel

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.

4.5.10 enum _clock_rootmux_ecspi_clk_sel

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.

4.5.11 enum _clock_rootmux_enet_axi_clk_sel

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.

4.5.12 enum _clock_rootmux_enet_ref_clk_sel

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.

4.5.13 enum _clock_rootmux_enet_timer_clk_sel

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.

4.5.14 enum _clock_rootmux_enet_phy_clk_sel

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.

4.5.15 enum _clock_rootmux_i2c_clk_sel

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.

4.5.16 enum _clock_rootmux_uart_clk_sel

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.

4.5.17 enum _clock_rootmux_gpt

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.

4.5.18 enum _clock_rootmux_wdog_clk_sel

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.

4.5.19 enum _clock_rootmux_pwm_clk_sel

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.

4.5.20 enum _clock_rootmux_sai_clk_sel

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.

4.5.21 enum _clock_rootmux_pdm_clk_sel

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.

4.5.22 enum _clock_rootmux_noc_clk_sel

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.

4.5.23 enum _clock_pll_gate

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.

4.5.24 enum _clock_gate_value

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.

4.5.25 enum _clock_pll_bypass_ctrl

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.

4.5.26 enum _ccm_analog_pll_clke

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.

4.5.27 anonymous enum

Enumerator

kANALOG_PlRefOsc24M reference OSC 24M
kANALOG_PlPadClk reference PAD CLK

4.6 Function Documentation

4.6.1 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 clock_root_control_t enumeration).
<i>mux</i>	Root mux value (see _ccm_rootmux_xxx enumeration).

4.6.2 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 clock_root_control_t enumeration).
----------------	--

Returns

Root mux value (see [_ccm_rootmux_xxx](#) enumeration).

4.6.3 static void CLOCK_EnableRoot([clock_root_control_t](#) *rootClk*) [inline], [static]

Parameters

<i>rootClk</i>	Root clock control (see clock_root_control_t enumeration)
----------------	---

4.6.4 static void CLOCK_DisableRoot([clock_root_control_t](#) *rootClk*) [inline], [static]

Parameters

<i>rootClk</i>	Root control (see clock_root_control_t enumeration)
----------------	---

4.6.5 static bool CLOCK_IsRootEnabled([clock_root_control_t](#) *rootClk*) [inline], [static]

Parameters

<i>rootClk</i>	Root control (see clock_root_control_t enumeration)
----------------	---

Returns

CCM root enabled or not.

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

4.6.6 void CLOCK_UpdateRoot([clock_root_control_t](#) *ccmRootClk*, [uint32_t](#) *mux*, [uint32_t](#) *pre*, [uint32_t](#) *post*)

Parameters

<i>ccmRootClk</i>	Root control (see clock_root_control_t enumeration)
<i>mux</i>	Root mux value (see _ccm_rootmux_xxx enumeration)
<i>pre</i>	Pre divider value (0-7, divider=n+1)
<i>post</i>	Post divider value (0-63, divider=n+1)

4.6.7 void CLOCK_SetRootDivider ([clock_root_control_t](#) *ccmRootClk*, [uint32_t](#) *pre*, [uint32_t](#) *post*)

Parameters

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

4.6.8 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 clock_root_control_t enumeration).
----------------	---

Returns

Root Pre divider value.

4.6.9 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 clock_root_control_t enumeration).
----------------	---

Returns

Root Post divider value.

4.6.10 static void CLOCK_ControlGate(*uintptr_t ccmGate*, *clock_gate_value_t control*) [inline], [static]

Parameters

<i>ccmGate</i>	Gate control (see clock_pll_gate_t and clock_ip_name_t enumeration)
<i>control</i>	Gate control value (see clock_gate_value_t)

4.6.11 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 clock_ip_name_t enumeration).
----------------	---

4.6.12 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 clock_ip_name_t enumeration).
----------------	---

4.6.13 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 clock_pll_ctrl_t enumeration)

4.6.14 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 clock_pll_ctrl_t enumeration)

4.6.15 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 ccm_analog_pll_control_t enumeration)
<i>bypass</i>	Bypass the PLL. <ul style="list-style-type: none"> • true: Bypass the PLL. • false: Do not bypass the PLL.

4.6.16 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 ccm_analog_pll_control_t enumeration)

Returns

- PLL bypass status.
- true: The PLL is bypassed.

- false: The PLL is not bypassed.

4.6.17 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 clock_pll_ctrl_t enumeration)

Returns

PLL lock status.

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

4.6.18 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 ccm_analog_pll_clock_t enumeration)

4.6.19 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 ccm_analog_pll_clock_t enumeration)

4.6.20 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 clock_pll_clke_t enumeration)
<i>override</i>	Override the PLL. <ul style="list-style-type: none">• true: Override the PLL clke, CCM will handle it.• false: Do not override the PLL clke.

4.6.21 static void CLOCK_OverridePIIPd (CCM_ANALOG_Type * *base*, [clock_pll_ctrl_t](#) *pdClock*, bool *override*) [inline], [static]

Parameters

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

4.6.22 void CLOCK_InitArmPll (const [ccm_analog_integer_pll_config_t](#) * *config*)

Parameters

<i>config</i>	Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).
---------------	--

Note

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

4.6.23 void CLOCK_InitSysPll1 (const [ccm_analog_integer_pll_config_t](#) * *config*)

Parameters

<i>config</i>	Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).
---------------	--

Note

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

4.6.24 void CLOCK_InitSysPLL2 (const ccm_analog_integer_pll_config_t * *config*)

Parameters

<i>config</i>	Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).
---------------	--

Note

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

4.6.25 void CLOCK_InitSysPLL3 (const ccm_analog_integer_pll_config_t * *config*)

Parameters

<i>config</i>	Pointer to the configuration structure(see ccm_analog_integer_pll_config_t enumeration).
---------------	--

Note

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

4.6.26 void CLOCK_InitAudioPLL1 (const ccm_analog_frac_pll_config_t * *config*)

Parameters

<i>config</i>	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).
---------------	---

Note

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

4.6.27 void CLOCK_InitAudioPII2 (const ccm_analog_frac_pll_config_t * *config*)

Parameters

<i>config</i>	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).
---------------	---

Note

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

4.6.28 void CLOCK_InitVideoPII1 (const ccm_analog_frac_pll_config_t * *config*)

Parameters

<i>config</i>	Pointer to the configuration structure(see ccm_analog_frac_pll_config_t enumeration).
---------------	---

4.6.29 void CLOCK_InitIntegerPII (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 ccm_analog_integer_pll_config_t enumeration).
<i>type</i>	integer pll type

4.6.30 **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

4.6.31 **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 ccm_analog_frac_pll_config_t enumeration).
<i>type</i>	fractional pll type.

4.6.32 **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

4.6.33 uint32_t CLOCK_GetPIIFreq (*clock_pll_ctrl_t pIIf*)

Parameters

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

Returns

Clock frequency

4.6.34 uint32_t CLOCK_GetPIIRefClkFreq (*clock_pll_ctrl_t ctrl*)

Parameters

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

Returns

Clock frequency

4.6.35 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

4.6.36 `uint32_t CLOCK_GetClockRootFreq (clock_root_t clockRoot)`

Parameters

<i>clockRoot</i>	The clock root used to get the frequency, please refer to clock_root_t .
------------------	--

Returns

The frequency of selected clock root.

4.6.37 `uint32_t CLOCK_GetCoreM4Freq (void)`

Returns

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

4.6.38 `uint32_t CLOCK_GetAxiFreq (void)`

Returns

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

4.6.39 `uint32_t CLOCK_GetAhbFreq (void)`

Returns

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

4.6.40 `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,

- #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, 0x303302C0
- #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, 0x303302C4
- #define **IOMUXC_GPIO1_IO13_USB1_OTG_OC** 0x3033005C, 0x1, 0x00000000, 0x0, 0x303302C4
- #define **IOMUXC_GPIO1_IO13_PWM2_OUT** 0x3033005C, 0x5, 0x00000000, 0x0, 0x303302C4
- #define **IOMUXC_GPIO1_IO14_GPIO1_IO14** 0x30330060, 0x0, 0x00000000, 0x0, 0x303302C8
- #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, 0x303302C8
- #define **IOMUXC_GPIO1_IO14_CCM_CLKO1** 0x30330060, 0x6, 0x00000000, 0x0, 0x303302C8
- #define **IOMUXC_GPIO1_IO15_GPIO1_IO15** 0x30330064, 0x0, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_GPIO1_IO15_USB2_OTG_OC** 0x30330064, 0x1, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_GPIO1_IO15_USDHC3_WP** 0x30330064, 0x4, 0x30330548, 0x2, 0x303302CC
- #define **IOMUXC_GPIO1_IO15_PWM4_OUT** 0x30330064, 0x5, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_GPIO1_IO15_CCM_CLKO2** 0x30330064, 0x6, 0x00000000, 0x0, 0x303302CC
- #define **IOMUXC_ENET_MDC_ENET1_MDC** 0x30330068, 0x0, 0x00000000, 0x0, 0x303302D0
- #define **IOMUXC_ENET_MDC_GPIO1_IO16** 0x30330068, 0x5, 0x00000000, 0x0, 0x303302D0
- #define **IOMUXC_ENET_MDIO_ENET1_MDIO** 0x3033006C, 0x0, 0x303304C0, 0x1, 0x303302D4
- #define **IOMUXC_ENET_MDIO_GPIO1_IO17** 0x3033006C, 0x5, 0x00000000, 0x0, 0x303302D4
- #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_RX_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, 0x3033035-
C
- #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
- #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, 0x303303D-
 - C
 - #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, 0x303303-
 - E4
 - #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_EVENTO** 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_EVENTI** 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, 0x3033040-
- C
- #define **IOMUXC_SAI1_TXD6_SRC_BOOT_CFG14** 0x303301A4, 0x6, 0x00000000, 0x0,
- 0x3033040C
- #define **IOMUXC_SAI1_TXD7_SAI1_TX_DATA7** 0x303301A8, 0x0, 0x00000000, 0x0,
- 0x30330410
- #define **IOMUXC_SAI1_TXD7_SAI6_MCLK** 0x303301A8, 0x1, 0x30330530, 0x1, 0x30330410
- #define **IOMUXC_SAI1_TXD7_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_RXC_SAI2_TX_BCLK** 0x303301C0, 0x0, 0x00000000, 0x0, 0x30330428
- #define **IOMUXC_SAI2_RXC_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_RX_SYNC** 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,

- 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_ECSPI1_SCLK_ECSPI1_SCLK** 0x303301F4, 0x0, 0x00000000, 0x0, 0x3033045C
 - #define **IOMUXC_ECSPI1_SCLK_UART3_RX** 0x303301F4, 0x1, 0x30330504, 0x0, 0x3033045-C
 - #define **IOMUXC_ECSPI1_SCLK_UART3_TX** 0x303301F4, 0x1, 0x00000000, 0X0, 0x3033045C
 - #define **IOMUXC_ECSPI1_SCLK_GPIO5_IO06** 0x303301F4, 0x5, 0x00000000, 0x0, 0x3033045C
 - #define **IOMUXC_ECSPI1_MOSI_ECSPI1_MOSI** 0x303301F8, 0x0, 0x00000000, 0x0, 0x30330460
 - #define **IOMUXC_ECSPI1_MOSI_UART3_TX** 0x303301F8, 0x1, 0x00000000, 0X0, 0x30330460
 - #define **IOMUXC_ECSPI1_MOSI_UART3_RX** 0x303301F8, 0x1, 0x30330504, 0x1, 0x30330460
 - #define **IOMUXC_ECSPI1_MOSI_GPIO5_IO07** 0x303301F8, 0x5, 0x00000000, 0x0, 0x30330460
 - #define **IOMUXC_ECSPI1_MISO_ECSPI1_MISO** 0x303301FC, 0x0, 0x00000000, 0x0,

- 0x30330464
- #define **IOMUXC_ECSP1_MISO_UART3_CTS_B** 0x303301FC, 0x1, 0x00000000, 0X0, 0x30330464
 - #define **IOMUXC_ECSP1_MISO_UART3_RTS_B** 0x303301FC, 0x1, 0x30330500, 0x0, 0x30330464
 - #define **IOMUXC_ECSP1_MISO_GPIO5_IO08** 0x303301FC, 0x5, 0x00000000, 0x0, 0x30330464
 - #define **IOMUXC_ECSP1_SS0_ECSP1_SS0** 0x30330200, 0x0, 0x00000000, 0x0, 0x30330468
 - #define **IOMUXC_ECSP1_SS0_UART3_RTS_B** 0x30330200, 0x1, 0x30330500, 0x1, 0x30330468
 - #define **IOMUXC_ECSP1_SS0_UART3_CTS_B** 0x30330200, 0x1, 0x00000000, 0X0, 0x30330468
 - #define **IOMUXC_ECSP1_SS0_GPIO5_IO09** 0x30330200, 0x5, 0x00000000, 0x0, 0x30330468
 - #define **IOMUXC_ECSP2_SCLK_ECSP2_SCLK** 0x30330204, 0x0, 0x00000000, 0x0, 0x3033046C
 - #define **IOMUXC_ECSP2_SCLK_UART4_RX** 0x30330204, 0x1, 0x3033050C, 0x0, 0x3033046C
 - #define **IOMUXC_ECSP2_SCLK_UART4_TX** 0x30330204, 0x1, 0x00000000, 0X0, 0x3033046C
 - #define **IOMUXC_ECSP2_SCLK_GPIO5_IO10** 0x30330204, 0x5, 0x00000000, 0x0, 0x3033046C
 - #define **IOMUXC_ECSP2_MOSI_ECSP2_MOSI** 0x30330208, 0x0, 0x00000000, 0x0, 0x30330470
 - #define **IOMUXC_ECSP2_MOSI_UART4_TX** 0x30330208, 0x1, 0x00000000, 0X0, 0x30330470
 - #define **IOMUXC_ECSP2_MOSI_UART4_RX** 0x30330208, 0x1, 0x3033050C, 0x1, 0x30330470
 - #define **IOMUXC_ECSP2_MOSI_GPIO5_IO11** 0x30330208, 0x5, 0x00000000, 0x0, 0x30330470
 - #define **IOMUXC_ECSP2_MISO_ECSP2_MISO** 0x3033020C, 0x0, 0x00000000, 0x0, 0x30330474
 - #define **IOMUXC_ECSP2_MISO_UART4_CTS_B** 0x3033020C, 0x1, 0x00000000, 0X0, 0x30330474
 - #define **IOMUXC_ECSP2_MISO_UART4_RTS_B** 0x3033020C, 0x1, 0x30330508, 0x0, 0x30330474
 - #define **IOMUXC_ECSP2_MISO_GPIO5_IO12** 0x3033020C, 0x5, 0x00000000, 0x0, 0x30330474
 - #define **IOMUXC_ECSP2_SS0_ECSP2_SS0** 0x30330210, 0x0, 0x00000000, 0x0, 0x30330478
 - #define **IOMUXC_ECSP2_SS0_UART4_RTS_B** 0x30330210, 0x1, 0x30330508, 0x1, 0x30330478
 - #define **IOMUXC_ECSP2_SS0_UART4_CTS_B** 0x30330210, 0x1, 0x00000000, 0X0, 0x30330478
 - #define **IOMUXC_ECSP2_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 ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))`
Computes the number of elements in an array.
- `#define SUPPRESS_FALL_THROUGH_WARNING()`
For switch case code block, if case section ends without "break;" statement, there will be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc.

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_LP SPI` = 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_SD SPI` = 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_EPC_USAR` = 57,
 `kStatusGroup_DMIC` = 58,
 `kStatusGroup_SDIF` = 59,
}

```

kStatusGroup_ELE = 167 }
    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.

Debug console type definition.

- #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

- `#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.

Min/max macros

- `#define MIN(a, b) (((a) < (b)) ? (a) : (b))`
Computes the minimum of a and b.
- `#define MAX(a, b) (((a) > (b)) ? (a) : (b))`
Computes the maximum of a and b.

UINT16_MAX(UINT32_MAX value

- `#define UINT16_MAX ((uint16_t)-1)`
Max value of uint16_t type.
- `#define UINT32_MAX ((uint32_t)-1)`
Max value of uint32_t type.

Atomic modification

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

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` is 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 protect 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)`
Add value val from the variable at address address.
- `#define SDK_ATOMIC_LOCAL_SUB(addr, val)`
Subtract value val to the variable at address address.
- `#define SDK_ATOMIC_LOCAL_SET(addr, bits)`
Set the bits specified by bits to the variable at address address.
- `#define SDK_ATOMIC_LOCAL_CLEAR(addr, bits)`
Clear the bits specified by bits to the variable at address address.
- `#define SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)`
Toggle the bits specified by bits to the variable at address address.
- `#define SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)`
For the variable at address address, clear the bits specified by clearBits and set the bits specified by 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_ALIGN(var, alignbytes) var __attribute__((aligned(alignbytes)))`
Macro to define a variable with alignbytes alignment.
- `#define SDK_L1DCACHE_ALIGN(var) SDK_ALIGN(var, FSL_FEATURE_L1DCACHE_LINESIZE_BYTEx)`
Macro to define a variable with L1 d-cache line size alignment.
- `#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.

Non-cacheable region definition macros

For initialized non-zero non-cacheable variables, please use "AT_NONCACHEABLE_SECTION_INIT(var) ={xx};" or "AT_NONCACHEABLE_SECTION_ALIGN_INIT(var) ={xx};" in your projects to

define them.

For zero-init non-cacheable variables, please use "AT_NONCACHEABLE_SECTION(var);" or "-AT_NONCACHEABLE_SECTION_ALIGN(var);" to define them, these zero-initied variables will be initialized to zero in system startup.

Note

For GCC, when the non-cacheable section is required, please define "__STARTUP_INITIALIZE_NONCACHEDATA" in your projects to make sure the non-cacheable section variables will be initialized in system startup.

- #define `AT_NONCACHEABLE_SECTION(var) __attribute__((section("NonCacheable,\"aw\",%nobits @)))` var
Define a variable var, and place it in non-cacheable section.
- #define `AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes) __attribute__((section("NonCacheable,\"aw\",%nobits @)))` var __attribute__((aligned(alignbytes)))
Define a variable var, and place it in non-cacheable section, the start address of the variable is aligned to alignbytes.
- #define `AT_NONCACHEABLE_SECTION_INIT(var) __attribute__((section("NonCacheable.-init")))` var
Define a variable var with initial value, and place it in non-cacheable section.
- #define `AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes) __attribute__((section("NonCacheable.init")))` var __attribute__((aligned(alignbytes)))
Define a variable var with initial value, and place it in non-cacheable section, the start address of the variable is aligned to alignbytes.

Time sensitive region

- #define `AT_QUICKACCESS_SECTION_CODE(func) __attribute__((section("CodeQuickAccess"), __noinline__))` func
Place function in a section which can be accessed quickly by core.
- #define `AT_QUICKACCESS_SECTION_DATA(var) __attribute__((section("DataQuickAccess")))` var
Place data in a section which can be accessed quickly by core.
- #define `AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes) __attribute__((section("DataQuickAccess")))` var __attribute__((aligned(alignbytes)))
Place data in a section which can be accessed quickly by core, and the variable address is set to align with alignbytes.

Ram Function

- #define `RAMFUNCTION_SECTION_CODE(func) __attribute__((section("RamFunction")))` func
Place function in ram.

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 MIN(a, b) (((a) < (b)) ? (a) : (b))
- 6.2.17 #define MAX(a, b) (((a) > (b)) ? (a) : (b))
- 6.2.18 #define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
- 6.2.19 #define UINT16_MAX ((uint16_t)-1)
- 6.2.20 #define UINT32_MAX ((uint32_t)-1)
- 6.2.21 #define SUPPRESS_FALL_THROUGH_WARNING()

To suppress this warning, "SUPPRESS_FALL_THROUGH_WARNING();" need to be added at the end of each case section which misses "break;" statement.

6.2.22 #define SDK_SIZEALIGN(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))

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_LP SPI Group number for LP SPI 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_SD SPI Group number for SD SPI 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_MIPI_DSI 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_SDIOSLV Group number for SDIOSLV 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_ELEMU Group number for ELEMU status codes.
kStatusGroup_QUEUEDSPI 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.

kStatusGroup_ELE Group number for ELE 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 DisableGlobalIRQ() .
----------------	--

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](#)
ECSPI user channel configure structure. [More...](#)
- struct [_ecspi_master_config](#)
ECSPI master configure structure. [More...](#)
- struct [_ecspi_slave_config](#)
ECSPI slave configure structure. [More...](#)
- struct [_ecspi_transfer](#)
ECSPI transfer structure. [More...](#)
- struct [_ecspi_master_handle](#)
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 enum **_ecspi_clock_polarity** **ecspi_clock_polarity_t**
ECSPI clock polarity configuration.
- typedef enum **_ecspi_clock_phase** **ecspi_clock_phase_t**
ECSPI clock phase configuration.
- typedef enum **_ecspi_data_ready** **ecspi_Data_ready_t**
ECSPI SPI_RDY signal configuration.
- typedef enum **_ecspi_channel_source** **ecspi_channel_source_t**
ECSPI channel select source.
- typedef enum **_ecspi_master_slave_mode** **ecspi_master_slave_mode_t**
ECSPI master or slave mode configuration.
- typedef enum **_ecspi_data_line_inactive_state_t** **ecspi_data_line_inactive_state_t**
ECSPI data line inactive state configuration.
- typedef enum **_ecspi_clock_inactive_state_t** **ecspi_clock_inactive_state_t**
ECSPI clock inactive state configuration.
- typedef enum **_ecspi_chip_select_active_state_t** **ecspi_chip_select_active_state_t**
ECSPI active state configuration.
- typedef enum **_ecspi_sample_period_clock_source** **ecspi_sample_period_clock_source_t**
ECSPI sample period clock configuration.
- typedef struct **_ecspi_channel_config** **ecspi_channel_config_t**
ECSPI user channel configure structure.
- typedef struct **_ecspi_master_config** **ecspi_master_config_t**
ECSPI master configure structure.
- typedef struct **_ecspi_slave_config** **ecspi_slave_config_t**
ECSPI slave configure structure.
- typedef struct **_ecspi_transfer** **ecspi_transfer_t**
ECSPI transfer structure.
- 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** {

kECSPI_PolarityActiveHigh = 0x0U,

kECSPI_PolarityActiveLow }

ECSPI clock polarity configuration.
- enum **_ecspi_clock_phase** {

kECSPI_ClockPhaseFirstEdge,

kECSPI_ClockPhaseSecondEdge }

ECSPI clock phase configuration.
- enum {

kECSPI_Tx fifoEmptyInterruptEnable = ECSPI_INTREG_TEEN_MASK,

kECSPI_TxFifoDataRequestInterruptEnable = ECSPI_INTREG_TDREN_MASK,

kECSPI_TxFifoFullInterruptEnable = ECSPI_INTREG_TFEN_MASK,

kECSPI_RxFifoReadyInterruptEnable = ECSPI_INTREG_RREN_MASK,

kECSPI_RxFifoDataRequestInterruptEnable = ECSPI_INTREG_RDREN_MASK,

kECSPI_RxFifoFullInterruptEnable = ECSPI_INTREG_RFEN_MASK,

kECSPI_RxFifoOverflowInterruptEnable = ECSPI_INTREG_ROEN_MASK,

kECSPI_TransferCompleteInterruptEnable = ECSPI_INTREG_TCEN_MASK,

kECSPI_AllInterruptEnable }

ECSPI interrupt sources.
- enum {

kECSPI_Tx fifoEmptyFlag = 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_M-

- ASK) }
- ECSPI DMA enable.*
- enum _ecspi_data_ready {
 kECSPI_DataReadyIgnore = 0x0U,
 kECSPI_DataReadyFallingEdge,
 kECSPI_DataReadyLowLevel }
- ECSPI SPI_RDY signal configuration.*
- enum _ecspi_channel_source {
 kECSPI_Channel0 = 0x0U,
 kECSPI_Channel1,
 kECSPI_Channel2,
 kECSPI_Channel3 }
- ECSPI channel select source.*
- enum _ecspi_master_slave_mode {
 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 {
 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

Data Fields

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

7.2.3.2 struct _ecspi_master_config

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::burstLength`

The length shall be less than 4096 bits

(2) `bool _ecspi_master_config::enableLoopback`

7.2.3.3 `struct _ecspi_slave_config`

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::burstLength`

The length shall be less than 4096 bits

7.2.3.4 `struct _ecspi_transfer`

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 Typedef Documentation

7.2.5.1 `typedef enum _ecspi_clock_polarity ecspi_clock_polarity_t`

7.2.5.2 `typedef enum _ecspi_clock_phase ecspi_clock_phase_t`

7.2.5.3 `typedef enum _ecspi_data_ready ecspi_Data_ready_t`

7.2.5.4 `typedef enum _ecspi_channel_source ecspi_channel_source_t`

7.2.5.5 `typedef enum _ecspi_master_slave_mode ecspi_master_slave_mode_t`

7.2.5.6 `typedef enum _ecspi_data_line_inactive_state_t ecspi_data_line_inactive_state_t`

7.2.5.7 `typedef enum _ecspi_clock_inactive_state_t ecspi_clock_inactive_state_t`

7.2.5.8 `typedef enum _ecspi_chip_select_active_state_t ecspi_chip_select_active_state_t`

7.2.5.9 `typedef enum _ecspi_sample_period_clock_source ecspi_sample_period_clock_source_t`

7.2.5.10 `typedef struct _ecspi_channel_config ecspi_channel_config_t`

7.2.5.11 `typedef struct _ecspi_master_config ecspi_master_config_t`

7.2.5.12 `typedef struct _ecspi_slave_config ecspi_slave_config_t`

7.2.6 Enumeration Type Documentation

7.2.6.1 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.

7.2.6.2 enum _ecspi_clock_polarity

Enumerator

kECSPI_PolarityActiveHigh Active-high ECSPI polarity high (idles low).
kECSPI_PolarityActiveLow Active-low ECSPI polarity low (idles high).

7.2.6.3 enum _ecspi_clock_phase

Enumerator

kECSPI_ClockPhaseFirstEdge First edge on SPSCK occurs at the middle of the first cycle of a data transfer.
kECSPI_ClockPhaseSecondEdge First edge on SPSCK occurs at the start of the first cycle of a data transfer.

7.2.6.4 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.

7.2.6.5 anonymous enum

Enumerator

kECSPI_Tx fifoEmptyFlag Transmit FIFO buffer empty flag.
kECSPI_TxFifoDataRequestFlag Transmit FIFO data request flag.
kECSPI_TxFifoFullFlag Transmit FIFO full flag.
kECSPI_RxFifoReadyFlag Receiver FIFO ready flag.
kECSPI_RxFifoDataRequestFlag Receiver FIFO data request flag.
kECSPI_RxFifoFullFlag Receiver FIFO full flag.

kECSPI_RxFifoOverFlowFlag Receiver FIFO buffer overflow flag.

kECSPI_TransferCompleteFlag Transfer complete flag.

7.2.6.6 anonymous enum

Enumerator

kECSPI_TxDmaEnable Tx DMA request source.

kECSPI_RxDmaEnable Rx DMA request source.

kECSPI_DmaAllEnable All DMA request source.

7.2.6.7 enum _ecspi_data_ready

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.

7.2.6.8 enum _ecspi_channel_source

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.

7.2.6.9 enum _ecspi_master_slave_mode

Enumerator

kECSPI_Slave ECSPI peripheral operates in slave mode.

kECSPI_Master ECSPI peripheral operates in master mode.

7.2.6.10 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.

7.2.6.11 enum _ecspi_clock_inactive_state_t

Enumerator

kECSPI_ClockInactiveStateLow The SCLK inactive state stays low.

kECSPI_ClockInactiveStateHigh The SCLK inactive state stays high.

7.2.6.12 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.

7.2.6.13 enum _ecspi_sample_period_clock_source

Enumerator

kECSPI_spiClock The sample period clock source is SCLK.

kECSPI_lowFreqClock The sample seriod clock source is low_frequency reference clock(32.768 kHz).

7.2.7 Function Documentation

7.2.7.1 uint32_t ECSPI_GetInstance (**EC SPI_Type * base**)

Parameters

<i>base</i>	EC SPI base address
-------------	---------------------

7.2.7.2 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
---------------	-----------------------------

7.2.7.3 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(ECSP10, &config);
```

Parameters

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

7.2.7.4 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
---------------	-----------------------------

7.2.7.5 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(ECSP11, &config);
```

Parameters

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

7.2.7.6 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
-------------	--------------------

7.2.7.7 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

7.2.7.8 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.

7.2.7.9 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 _ecspi_flags could get the related status.

7.2.7.10 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"> • kECSPI_Tx_fifoEmptyInterruptEnable • kECSPI_Tx_FifoDataRequrstInterruptEnable • kECSPI_Tx_FifoFullInterruptEnable • kECSPI_Rx_FifoReadyInterruptEnable • kECSPI_Rx_FifoDataRequrstInterruptEnable • kECSPI_Rx_FifoFullInterruptEnable • kECSPI_Rx_FifoOverFlowInterruptEnable • kECSPI_TransferCompleteInterruptEnable • kECSPI_AllInterruptEnable

**7.2.7.11 static void ECSPI_DisableInterrupts (*ECSPI_Type* * *base*, *uint32_t* *mask*)
[inline], [static]**

Parameters

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

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

Parameters

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

7.2.7.13 static bool ECSPI_IsMaster (*EC SPI_Type* * *base*, *ecspi_channel_source_t* *channel*) [inline], [static]

Parameters

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

Returns

mode of channel

7.2.7.14 static void ECSPI_EnableDMA (*EC SPI_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">• kECSPI_TxDmaEnable• kECSPI_RxDmaEnable• kECSPI_DmaAllEnable
<i>enable</i>	True means enable DMA, false means disable DMA

7.2.7.15 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.

7.2.7.16 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.

7.2.7.17 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.

7.2.7.18 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

7.2.7.19 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.

7.2.7.20 **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.

7.2.7.21 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.

7.2.7.22 static uint32_t ECSPI_ReadData (**ECSPI_Type** * *base*) [inline], [static]

Parameters

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

Returns

Data in the register.

7.2.7.23 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.

7.2.7.24 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_ECSPI_Timeout</i>	The transfer timed out and was aborted.

7.2.7.25 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 ecspi_transfer_t 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.

7.2.7.26 **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.

7.2.7.27 **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.

7.2.7.28 **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.

**7.2.7.29 void ECSPI_SlaveTransferCreateHandle (`ECSPI_Type` * *base*,
`ecspi_slave_handle_t` * *handle*, `ecspi_slave_callback_t` *callback*, `void` * *userData*)**

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

Parameters

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

**7.2.7.30 static status_t ECSPI_SlaveTransferNonBlocking (`ECSPI_Type` * *base*,
`ecspi_slave_handle_t` * *handle*, `ecspi_transfer_t` * *xfer*) [inline], [static]**

Note

The API returns immediately after the transfer initialization is finished.

Parameters

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

Return values

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

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

Parameters

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

Return values

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

7.2.7.32 **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.

7.2.7.33 **void ECSPI_SlaveTransferHandleIRQ (*ECSPI_Type* * *base*, *ecspi_slave_handle_t* * *handle*)**

Parameters

<i>base</i>	ECSPI peripheral base address.
<i>handle</i>	pointer to <i>ecspi_slave_handle_t</i> 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

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

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>	ECSPI is not idle, is running another transfer.

7.4.5.5 void ECSPI_MasterTransferAbortSDMA (**ECSPI_Type** * *base*, **ecspi_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*, **ecspi_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 uninitialized 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 uninitialize */
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 uninitialize */
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 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`
Defines the receive buffer descriptor structure for the little endian system. [More...](#)
- struct `_enet_tx_bd_struct`
Defines the enhanced transmit buffer descriptor structure for the little endian system. [More...](#)
- struct `_enet_data_error_stats`
Defines the ENET data error statistics structure. [More...](#)
- struct `_enet_rx_frame_error`
Defines the Rx frame error structure. [More...](#)
- struct `_enet_transfer_stats`
Defines the ENET transfer statistics structure. [More...](#)
- struct `enet_frame_info`
Defines the frame info structure. [More...](#)

- struct `_enet_tx_dirty_ring`
Defines the ENET transmit dirty addresses ring/queue structure. [More...](#)
- struct `_enet_buffer_config`
Defines the receive buffer descriptor configuration structure. [More...](#)
- struct `_enet_intcoalesce_config`
Defines the interrupt coalescing configure structure. [More...](#)
- struct `_enet_avb_config`
Defines the ENET AVB Configure structure. [More...](#)
- struct `_enet_config`
Defines the basic configuration structure for the ENET device. [More...](#)
- struct `_enet_tx_bd_ring`
Defines the ENET transmit buffer descriptor ring/queue structure. [More...](#)
- struct `_enet_rx_bd_ring`
Defines the ENET receive buffer descriptor ring/queue structure. [More...](#)
- struct `_enet_handle`
Defines the ENET handler structure. [More...](#)

Macros

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

Typedefs

- typedef enum `_enet_mii_mode` `enet_mii_mode_t`
Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.
- typedef enum `_enet_mii_speed` `enet_mii_speed_t`
Defines the 10/100/1000 Mbps speed for the MII data interface.
- typedef enum `_enet_mii_duplex` `enet_mii_duplex_t`
Defines the half or full duplex for the MII data interface.
- typedef enum `_enet_mii_write` `enet_mii_write_t`
Define the MII opcode for normal MDIO_CLAUSES_22 Frame.
- typedef enum `_enet_mii_read` `enet_mii_read_t`
Defines the read operation for the MII management frame.
- typedef enum
`_enet_mii_extend_opcode` `enet_mii_extend_opcode`
Define the MII opcode for extended MDIO_CLAUSES_45 Frame.
- typedef enum
`_enet_special_control_flag` `enet_special_control_flag_t`
Defines a special configuration for ENET MAC controller.
- typedef enum `_enet_interrupt_enable` `enet_interrupt_enable_t`
List of interrupts supported by the peripheral.
- typedef enum `_enet_event` `enet_event_t`
Defines the common interrupt event for callback use.
- typedef enum `_enet_idle_slope` `enet_idle_slope_t`
Defines certain idle slope for bandwidth fraction.
- typedef enum `_enet_tx_accelerator` `enet_tx_accelerator_t`
Defines the transmit accelerator configuration.
- typedef enum `_enet_rx_accelerator` `enet_rx_accelerator_t`
Defines the receive accelerator configuration.
- typedef struct `_enet_rx_bd_struct` `enet_rx_bd_struct_t`

- *Defines the receive buffer descriptor structure for the little endian system.*
- **typedef struct _enet_tx_bd_struct enet_tx_bd_struct_t**
Defines the enhanced transmit buffer descriptor structure for the little endian system.
- **typedef struct _enet_data_error_stats enet_data_error_stats_t**
Defines the ENET data error statistics structure.
- **typedef struct _enet_rx_frame_error enet_rx_frame_error_t**
Defines the Rx frame error structure.
- **typedef struct _enet_transfer_stats enet_transfer_stats_t**
Defines the ENET transfer statistics structure.
- **typedef struct enet_frame_info enet_frame_info_t**
Defines the frame info structure.
- **typedef struct _enet_tx_dirty_ring enet_tx_dirty_ring_t**
Defines the ENET transmit dirty addresses ring/queue structure.
- **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 struct _enet_buffer_config enet_buffer_config_t**
Defines the receive buffer descriptor configuration structure.
- **typedef struct _enet_intcoalesce_config enet_intcoalesce_config_t**
Defines the interrupt coalescing configure structure.
- **typedef struct _enet_avb_config enet_avb_config_t**
Defines the ENET AVB Configure structure.
- **typedef void(*enet_callback_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)**
ENET callback function.
- **typedef struct _enet_config enet_config_t**
Defines the basic configuration structure for the ENET device.
- **typedef struct _enet_tx_bd_ring enet_tx_bd_ring_t**
Defines the ENET transmit buffer descriptor ring/queue structure.
- **typedef struct _enet_rx_bd_ring enet_rx_bd_ring_t**
Defines the ENET receive buffer descriptor ring/queue structure.
- **typedef void(*enet_isr_ring_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId)**
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 {

kENET_MiiMode = 0U,

kENET_RmiiMode = 1U,

kENET_RgmiiMode = 2U }

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.
- enum _enet_mii_speed {

kENET_MiiSpeed10M = 0U,

kENET_MiiSpeed100M = 1U,

kENET_MiiSpeed1000M = 2U }

Defines the 10/100/1000 Mbps speed for the MII data interface.
- enum _enet_mii_duplex {

kENET_MiiHalfDuplex = 0U,

kENET_MiiFullDuplex }

Defines the half or full duplex for the MII data interface.
- enum _enet_mii_write {

kENET_MiiWriteNoCompliant = 0U,

kENET_MiiWriteValidFrame }

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.
- enum _enet_mii_read {

kENET_MiiReadValidFrame = 2U,

kENET_MiiReadNoCompliant = 3U }

Defines the read operation for the MII management frame.
- enum _enet_mii_extend_opcode {

kENET_MiiAddrWrite_C45 = 0U,

kENET_MiiWriteFrame_C45 = 1U,

kENET_MiiReadFrame_C45 = 3U }

Define the MII opcode for extended MDIO_CLAUSES_45 Frame.
- enum _enet_special_control_flag {

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,

kENET_ControlSVLANEnable = 0x0400U,

kENET_ControlVLANUseSecondTag = 0x0800U }

Defines a special configuration for ENET MAC controller.
- enum _enet_interrupt_enable {

```

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_RxFlush2Interrupt = ENET_EIR_RXFLUSH_2_MASK,
kENET_RxFlush1Interrupt = ENET_EIR_RXFLUSH_1_MASK,
kENET_RxFlush0Interrupt = ENET_EIR_RXFLUSH_0_MASK,
kENET_TxFrame2Interrupt = ENET_EIR_TXF2_MASK,
kENET_TxBuffer2Interrupt = ENET_EIR_TXB2_MASK,
kENET_RxFrame2Interrupt = ENET_EIR_RXF2_MASK,
kENET_RxBuffer2Interrupt = ENET_EIR_RXB2_MASK,
kENET_TxFrame1Interrupt = ENET_EIR_TXF1_MASK,
kENET_TxBuffer1Interrupt = ENET_EIR_TXB1_MASK,
kENET_RxFrame1Interrupt = ENET_EIR_RXF1_MASK,
kENET_RxBuffer1Interrupt = ENET_EIR_RXB1_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` {

kENET_RxEvent,

kENET_TxEvent,

kENET_ErrEvent,

kENET_WakeUpEvent,

kENET_TimeStampEvent,

kENET_TimeStampAvailEvent }

Defines the common interrupt event for callback use.

- enum `_enet_idle_slope` {

```

kENET_IdleSlope1 = 1U,
kENET_IdleSlope2 = 2U,
kENET_IdleSlope4 = 4U,
kENET_IdleSlope8 = 8U,
kENET_IdleSlope16 = 16U,
kENET_IdleSlope32 = 32U,
kENET_IdleSlope64 = 64U,
kENET_IdleSlope128 = 128U,
kENET_IdleSlope256 = 256U,
kENET_IdleSlope384 = 384U,
kENET_IdleSlope512 = 512U,
kENET_IdleSlope640 = 640U,
kENET_IdleSlope768 = 768U,
kENET_IdleSlope896 = 896U,
kENET_IdleSlope1024 = 1024U,
kENET_IdleSlope1152 = 1152U,
kENET_IdleSlope1280 = 1280U,
kENET_IdleSlope1408 = 1408U,
kENET_IdleSlope1536 = 1536U }

```

Defines certain idle slope for bandwidth fraction.

- enum `_enet_tx_accelerator` {

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` {

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, 1))`
Defines the driver version.

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

Defines the queue number.

- #define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U
Empty bit mask.
- #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U
Software owner one mask.
- #define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U
Next buffer descriptor is the start address.
- #define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_MASK 0x1000U
Software owner two mask.
- #define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U
Last BD of the frame mask.
- #define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U
Received because of the promiscuous mode.
- #define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U
Broadcast packet mask.
- #define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U
Multicast packet mask.
- #define ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK 0x0020U
Length violation mask.
- #define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U
Non-octet aligned frame mask.
- #define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U
CRC error mask.
- #define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U
FIFO overrun mask.
- #define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U
Frame is truncated mask.

Control and status bit masks of the transmit buffer descriptor.

- #define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U
Ready bit mask.
- #define ENET_BUFFDESCRIPTOR_TX_SOFTWENER1_MASK 0x4000U
Software owner one mask.
- #define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U
Wrap buffer descriptor mask.
- #define ENET_BUFFDESCRIPTOR_TX_SOFTWENER2_MASK 0x1000U
Software owner two mask.
- #define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U
Last BD of the frame mask.
- #define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_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
Enet Tx interrupt flag.
- #define ENET_RX_INTERRUPT
Enet Rx interrupt flag.
- #define ENET_TS_INTERRUPT ((uint32_t)kENET_TsTimerInterrupt | (uint32_t)kENET_TsAvailInterrupt)
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)

- 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 write command.
- static void [ENET_MDIOWrite](#) (ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t data)

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_MDIOWrite](#) (ENET_Type *base, uint8_t phyAddr, uint8_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 22 format.
- static void [ENET_StartExtC45SMIWriteReg](#) (ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr)

Sends the MDIO IEEE802.3 Clause 45 format write register command.
- static void [ENET_StartExtC45SMIWriteData](#) (ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t data)

Sends the MDIO IEEE802.3 Clause 45 format write data command.
- static void [ENET_StartExtC45SMIReadData](#) (ENET_Type *base, uint8_t portAddr, uint8_t devAddr)

Sends the MDIO IEEE802.3 Clause 45 format read data command.
- status_t [ENET_MDIOC45Write](#) (ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t data)

MDIO write with IEEE802.3 Clause 45 format.
- status_t [ENET_MDIOC45Read](#) (ENET_Type *base, uint8_t portAddr, uint8_t devAddr, uint16_t regAddr, uint16_t *pData)

MDIO read with IEEE802.3 Clause 45 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_ring_t ISRHandler)
Set the second level Rx IRQ handler.
- void `ENET_SetTxISRHandler` (ENET_Type *base, enet_isr_ring_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, uint32_t ringId)
The transmit IRQ handler.
- void `ENET_ReceiveIRQHandler` (ENET_Type *base, enet_handle_t *handle, uint32_t ringId)
The receive IRQ handler.
- void `ENET_CommonFrame1IRQHandler` (ENET_Type *base)
the common IRQ handler for the tx/rx irq handler.
- void `ENET_CommonFrame2IRQHandler` (ENET_Type *base)
the common IRQ handler for the tx/rx 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

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

8.4.2 struct _enet_tx_bd_struct

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

8.4.3 struct _enet_data_error_stats

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

8.4.4 struct _enet_rx_frame_error

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

8.4.5 struct _enet_transfer_stats

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 statsTxCrcAlignErr`
The transmit frame is error.
- `uint32_t statsTxFifoUnderRunErr`
Tx FIFO underrun count.

Field Documentation

- (1) `uint32_t _enet_transfer_stats::statsRxFrameCount`
- (2) `uint32_t _enet_transfer_stats::statsRxFrameOk`
- (3) `uint32_t _enet_transfer_stats::statsRxCrcErr`
- (4) `uint32_t _enet_transfer_stats::statsRxAlignErr`
- (5) `uint32_t _enet_transfer_stats::statsRxDropInvalidSFD`
- (6) `uint32_t _enet_transfer_stats::statsRxFifoOverflowErr`
- (7) `uint32_t _enet_transfer_stats::statsTxFrameCount`
- (8) `uint32_t _enet_transfer_stats::statsTxFrameOk`
- (9) `uint32_t _enet_transfer_stats::statsTxCrcAlignErr`
- (10) `uint32_t _enet_transfer_stats::statsTxFifoUnderRunErr`

8.4.6 struct enet_frame_info**Data Fields**

- `void * context`
User specified data.

8.4.7 struct _enet_tx_dirty_ring**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::txDirtyBase`
- (2) `uint16_t _enet_tx_dirty_ring::txGenIdx`
- (3) `uint16_t _enet_tx_dirty_ring::txConsumIdx`
- (4) `uint16_t _enet_tx_dirty_ring::txRingLen`
- (5) `bool _enet_tx_dirty_ring::isFull`

8.4.8 struct _enet_buffer_config

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

8.4.9 struct _enet_intcoalesce_config

Data Fields

- `uint8_t txCoalesceFrameCount [FSL_FEATURE_ENET_QUEUE]`
Transmit interrupt coalescing frame count threshold.
- `uint16_t txCoalesceTimeCount [FSL_FEATURE_ENET_QUEUE]`
Transmit interrupt coalescing timer count threshold.
- `uint8_t rxCoalesceFrameCount [FSL_FEATURE_ENET_QUEUE]`
Receive interrupt coalescing frame count threshold.
- `uint16_t rxCoalesceTimeCount [FSL_FEATURE_ENET_QUEUE]`
Receive interrupt coalescing timer count threshold.

Field Documentation

- (1) `uint8_t _enet_intcoalesce_config::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- (2) `uint16_t _enet_intcoalesce_config::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`
- (3) `uint8_t _enet_intcoalesce_config::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- (4) `uint16_t _enet_intcoalesce_config::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

8.4.10 struct_enet_avb_config

This is used for to configure the extended ring 1 and ring 2.

1. The classification match format is $(\text{CMP3} \ll 12) | (\text{CMP2} \ll 8) | (\text{CMP1} \ll 4) | \text{CMP0}$. composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

1. The idleSlope is used to calculate the Band Width fraction, $\text{BW fraction} = 1 / (1 + 512/\text{idleSlope})$. For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

Data Fields

- `uint16_t rxClassifyMatch [FSL_FEATURE_ENET_QUEUE-1]`
The classification match value for the ring.
- `enet_idle_slope_t idleSlope [FSL_FEATURE_ENET_QUEUE-1]`
The idle slope for certian bandwidth fraction.

Field Documentation

- (1) `uint16_t _enet_avb_config::rxClassifyMatch[FSL_FEATURE_ENET_QUEUE-1]`
- (2) `enet_idle_slope_t _enet_avb_config::idleSlope[FSL_FEATURE_ENET_QUEUE-1]`

8.4.11 struct_enet_config

Note:

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

- words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.
4. When "kENET_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
 5. When "kENET_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
 6. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.
 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 rxFifoStatEmptyThreshold**
For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept.
- **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.
- **enet_intcoalesce_config_t * intCoalesceCfg**
If the interrupt coalescence is not required in the ring n(0,1,2),

- **uint8_t ringNum**
please set to NULL.
- **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::macSpecialConfig**

A logical OR of "enet_special_control_flag_t".

(2) **uint32_t _enet_config::interrupt**

A logical OR of "enet_interrupt_enable_t".

(3) **uint16_t _enet_config::rxMaxFrameLen**

(4) **enet_mii_mode_t _enet_config::miiMode**

(5) **enet_mii_speed_t _enet_config::miiSpeed**

(6) **enet_mii_duplex_t _enet_config::miiDuplex**

(7) **uint8_t _enet_config::rxAccelerConfig**

(8) **uint8_t _enet_config::txAccelerConfig**

(9) **uint16_t _enet_config::pauseDuration**

(10) **uint8_t _enet_config::rxFifoEmptyThreshold**

(11) **uint8_t _enet_config::rxFifoStatEmptyThreshold**

If the limit is reached, reception continues and a pause frame is triggered.

(12) **uint8_t _enet_config::rxFifoFullThreshold**

(13) **uint8_t _enet_config::txFifoWatermark**

(14) **enet_intcoalesce_config_t* _enet_config::intCoalesceCfg**

(15) **uint8_t _enet_config::ringNum**

default with 1 – single ring.

- (16) `enet_rx_alloc_callback_t _enet_config::rxBuffAlloc`
- (17) `enet_rx_free_callback_t _enet_config::rxBuffFree`
- (18) `enet_callback_t _enet_config::callback`
- (19) `void* _enet_config::userData`

8.4.12 struct _enet_tx_bd_ring

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 txConsumIdx`
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::txBdBase`
- (2) `uint16_t _enet_tx_bd_ring::txGenIdx`
- (3) `uint16_t _enet_tx_bd_ring::txConsumIdx`
- (4) `volatile uint16_t _enet_tx_bd_ring::txDescUsed`
- (5) `uint16_t _enet_tx_bd_ring::txRingLen`

8.4.13 struct _enet_rx_bd_ring

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::rxBdBase
- (2) uint16_t _enet_rx_bd_ring::rxGenIdx
- (3) uint16_t _enet_rx_bd_ring::rxRingLen

8.4.14 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::rxBdRing[FSL_FEATURE_ENET_QUEUE]
- (2) enet_tx_bd_ring_t _enet_handle::txBdRing[FSL_FEATURE_ENET_QUEUE]
- (3) uint16_t _enet_handle::rxBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]
- (4) uint16_t _enet_handle::txBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]
- (5) bool _enet_handle::rxMaintainEnable[FSL_FEATURE_ENET_QUEUE]
- (6) bool _enet_handle::txMaintainEnable[FSL_FEATURE_ENET_QUEUE]
- (7) uint8_t _enet_handle::ringNum
- (8) enet_callback_t _enet_handle::callback
- (9) void* _enet_handle::userData
- (10) enet_tx_dirty_ring_t _enet_handle::txDirtyRing[FSL_FEATURE_ENET_QUEUE]
- (11) bool _enet_handle::txReclaimEnable[FSL_FEATURE_ENET_QUEUE]
- (12) enet_rx_alloc_callback_t _enet_handle::rxBuffAlloc
- (13) enet_rx_free_callback_t _enet_handle::rxBuffFree

8.5 Macro Definition Documentation

- 8.5.1 `#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 7, 1))`
- 8.5.2 `#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U`
- 8.5.3 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U`
- 8.5.4 `#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U`
- 8.5.5 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U`
- 8.5.6 `#define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U`
- 8.5.7 `#define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U`
- 8.5.8 `#define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U`
- 8.5.9 `#define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U`
- 8.5.10 `#define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK 0x0020U`
- 8.5.11 `#define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U`
- 8.5.12 `#define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U`
- 8.5.13 `#define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U`
- 8.5.14 `#define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U`
- 8.5.15 `#define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U`
- 8.5.16 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK 0x4000U`
- 8.5.17 `#define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U`
- 8.5.18 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK 0x1000U`
- 8.5.19 `#define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U`
- 8.5.20 `#define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U`
- 8.5.21 `#define ENET_BUFFDESCRIPTOR_RX_ERR_MASK`

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

8.5.22 #define ENET_FRAME_MAX_FRAMELEN 1518U

8.5.23 #define ENET_FRAME_VLAN_TAGLEN 4U

8.5.24 #define ENET_FRAME_CRC_LEN 4U

8.5.25 #define ENET_FIFO_MIN_RX_FULL 5U

8.5.26 #define ENET_RX_MIN_BUFFERSIZE 256U

8.5.27 #define ENET_PHY_MAXADDRESS (ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT)

8.5.28 #define ENET_TX_INTERRUPT

Value:

```
((uint32_t)kENET_TxFrameInterrupt | (uint32_t)
 kENET_TxBufferInterrupt | (uint32_t)
 kENET_TxFramelInterrupt | \
 (uint32_t)kENET_TxBuffer1Interrupt | (uint32_t)
 kENET_TxFrame2Interrupt | \
 (uint32_t)kENET_TxBuffer2Interrupt)
```

8.5.29 #define ENET_RX_INTERRUPT

Value:

```
((uint32_t)kENET_RxFrameInterrupt | (uint32_t)
 kENET_RxBufferInterrupt | (uint32_t)
 kENET_RxFramelInterrupt | \
 (uint32_t)kENET_RxBuffer1Interrupt | (uint32_t)
 kENET_RxFrame2Interrupt | \
 (uint32_t)kENET_RxBuffer2Interrupt)
```

8.5.30 #define ENET_TS_INTERRUPT ((uint32_t)kENET_TsTimerInterrupt | (uint32_t)kENET_TsAvailInterrupt)

8.5.31 #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 enum _enet_mii_mode enet_mii_mode_t

8.6.2 typedef enum _enet_mii_speed enet_mii_speed_t

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

8.6.3 typedef enum _enet_mii_duplex enet_mii_duplex_t

8.6.4 typedef enum _enet_mii_write enet_mii_write_t

8.6.5 typedef enum _enet_mii_read enet_mii_read_t

8.6.6 typedef enum _enet_mii_extend_opcode enet_mii_extend_opcode

8.6.7 typedef enum _enet_special_control_flag 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.

8.6.8 typedef enum _enet_interrupt_enable 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.

- 8.6.9 `typedef enum _enet_event enet_event_t`**
- 8.6.10 `typedef enum _enet_idle_slope enet_idle_slope_t`**
- 8.6.11 `typedef enum _enet_tx_accelerator enet_tx_accelerator_t`**
- 8.6.12 `typedef enum _enet_rx_accelerator enet_rx_accelerator_t`**
- 8.6.13 `typedef struct _enet_rx_bd_struct enet_rx_bd_struct_t`**
- 8.6.14 `typedef struct _enet_tx_bd_struct enet_tx_bd_struct_t`**
- 8.6.15 `typedef struct _enet_data_error_stats enet_data_error_stats_t`**
- 8.6.16 `typedef struct _enet_rx_frame_error enet_rx_frame_error_t`**
- 8.6.17 `typedef struct _enet_transfer_stats enet_transfer_stats_t`**
- 8.6.18 `typedef struct enet_frame_info enet_frame_info_t`**
- 8.6.19 `typedef struct _enet_tx_dirty_ring enet_tx_dirty_ring_t`**
- 8.6.20 `typedef void*(* enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)`**
- 8.6.21 `typedef void(* enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)`**
- 8.6.22 `typedef struct _enet_buffer_config 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.`

8.6.23 `typedef struct _enet_intcoalesce_config enet_intcoalesce_config_t`

8.6.24 `typedef struct _enet_avb_config enet_avb_config_t`

This is used for to configure the extended ring 1 and ring 2.

1. The classification match format is `(CMP3 << 12) | (CMP2 << 8) | (CMP1 << 4) | CMP0`. composed of four 3-bit compared VLAN priority field `cmp0~cmp3`, `cm0 ~ cmp3` are used in parallel.

If `CMP1,2,3` are not unused, please set them to the same value as `CMP0`.

1. The `idleSlope` is used to calculate the Band Width fraction, $BW\ fraction = 1 / (1 + 512/idleSlope)$. For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

8.6.25 `typedef void(* enet_callback_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)`

8.6.26 `typedef struct _enet_config enet_config_t`

Note:

1. `macSpecialConfig` is used for a special control configuration, A logical OR of "`enet_special_control_flag_t`". For a special configuration for MAC, set this parameter to 0.
2. `txWatermark` is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO 3 - 192 bytes written to TX FIFO The maximum of `txWatermark` is 0x2F - 4032 bytes written to TX FIFO `txWatermark` allows minimizing the transmit latency to set the `txWatermark` to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
3. `rxFifoFullThreshold` is similar to the `txWatermark` for cut-through operation in RX. It is in 64-bit words. The minimum is `ENET_FIFO_MIN_RX_FULL` and the maximum is `0xFF`. If the end of the frame is stored in FIFO and the frame size if smaller than the `txWatermark`, the frame is still transmitted. The rule is the same for `rxFifoFullThreshold` in the receive direction.
4. When "`kENET_ControlFlowControlEnable`" is set in the `macSpecialConfig`, ensure that the `pauseDuration`, `rxFifoEmptyThreshold`, and `rxFifoStatEmptyThreshold` are set for flow control enabled case.
5. When "`kENET_ControlStoreAndFwdDisabled`" is set in the `macSpecialConfig`, ensure that the `rxFifoFullThreshold` and `txFifoWatermark` are set for store and forward disable.

6. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.
7. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

8.6.27 `typedef struct _enet_tx_bd_ring enet_tx_bd_ring_t`

8.6.28 `typedef struct _enet_rx_bd_ring enet_rx_bd_ring_t`

8.6.29 `typedef void(* enet_isr_ring_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId)`

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`

Enumerator

kENET_MiiMode MII mode for data interface.

kENET_RmiiMode RMII mode for data interface.

kENET_RgmiiMode RGMII mode for data interface.

8.7.3 `enum _enet_mii_speed`

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

Enumerator

kENET_MiiSpeed10M Speed 10 Mbps.

kENET_MiiSpeed100M Speed 100 Mbps.
kENET_MiiSpeed1000M Speed 1000M bps.

8.7.4 enum _enet_mii_duplex

Enumerator

kENET_MiiHalfDuplex Half duplex mode.
kENET_MiiFullDuplex Full duplex mode.

8.7.5 enum _enet_mii_write

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

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_mii_extend_opcode

Enumerator

kENET_MiiAddrWrite_C45 Address Write operation.
kENET_MiiWriteFrame_C45 Write frame operation for a valid MII management frame.
kENET_MiiReadFrame_C45 Read frame operation for a valid MII management frame.

8.7.8 enum _enet_special_control_flag

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).
- kENET_ControlSVLANEnable*** Enable S-VLAN.
- kENET_ControlVLANUseSecondTag*** Enable extracting the second vlan tag for further processing.

8.7.9 enum _enet_interrupt_enable

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_RxFlush2Interrupt*** Rx DMA ring2 flush indication.
- kENET_RxFlush1Interrupt*** Rx DMA ring1 flush indication.
- kENET_RxFlush0Interrupt*** RX DMA ring0 flush indication.
- kENET_TxFrame2Interrupt*** Tx frame interrupt for Tx ring/class 2.
- kENET_TxBuffer2Interrupt*** Tx buffer interrupt for Tx ring/class 2.
- kENET_RxFrame2Interrupt*** Rx frame interrupt for Rx ring/class 2.
- kENET_RxBuffer2Interrupt*** Rx buffer interrupt for Rx ring/class 2.
- kENET_TxFrame1Interrupt*** Tx frame interrupt for Tx ring/class 1.
- kENET_TxBuffer1Interrupt*** Tx buffer interrupt for Tx ring/class 1.

kENET_RxFrame1Interrupt Rx frame interrupt for Rx ring/class 1.

kENET_RxBuffer1Interrupt Rx buffer interrupt for Rx ring/class 1.

kENET_TsAvailInterrupt TS AVAIL interrupt source for PTP.

kENET_TsTimerInterrupt TS WRAP interrupt source for PTP.

8.7.10 enum _enet_event

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.

8.7.11 enum _enet_idle_slope

Enumerator

kENET_IdleSlope1 The bandwidth fraction is about 0.002.

kENET_IdleSlope2 The bandwidth fraction is about 0.003.

kENET_IdleSlope4 The bandwidth fraction is about 0.008.

kENET_IdleSlope8 The bandwidth fraction is about 0.02.

kENET_IdleSlope16 The bandwidth fraction is about 0.03.

kENET_IdleSlope32 The bandwidth fraction is about 0.06.

kENET_IdleSlope64 The bandwidth fraction is about 0.11.

kENET_IdleSlope128 The bandwidth fraction is about 0.20.

kENET_IdleSlope256 The bandwidth fraction is about 0.33.

kENET_IdleSlope384 The bandwidth fraction is about 0.43.

kENET_IdleSlope512 The bandwidth fraction is about 0.50.

kENET_IdleSlope640 The bandwidth fraction is about 0.56.

kENET_IdleSlope768 The bandwidth fraction is about 0.60.

kENET_IdleSlope896 The bandwidth fraction is about 0.64.

kENET_IdleSlope1024 The bandwidth fraction is about 0.67.

kENET_IdleSlope1152 The bandwidth fraction is about 0.69.

kENET_IdleSlope1280 The bandwidth fraction is about 0.71.

kENET_IdleSlope1408 The bandwidth fraction is about 0.73.

kENET_IdleSlope1536 The bandwidth fraction is about 0.75.

8.7.12 enum _enet_tx_accelerator

Enumerator

- kENET_TxAccelIsShift16Enabled*** Transmit FIFO shift-16.
- kENET_TxAccelIpCheckEnabled*** Insert IP header checksum.
- kENET_TxAccelProtoCheckEnabled*** Insert protocol checksum.

8.7.13 enum _enet_rx_accelerator

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>isPreambleDisabled</i>	<p>The preamble disable flag.</p> <ul style="list-style-type: none"> • true Enables the preamble. • false Disables the preamble.

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_MDIORRead\(\)](#) 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_MDIORRead (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 static void ENET_StartExtC45SMIWriteReg (ENET_Type * *base*, uint8_t *portAddr*, uint8_t *devAddr*, uint16_t *regAddr*) [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_MDIOC45Write\(\)](#)/[ENET_MDIOC45Read\(\)](#) can be called. For

customized requirements, implement with combining separated APIs.

Parameters

<i>base</i>	ENET peripheral base address.
<i>portAddr</i>	The MDIO port address(PHY address).
<i>devAddr</i>	The device address.
<i>regAddr</i>	The PHY register address.

8.8.17 static void ENET_StartExtC45SMIWriteData (ENET_Type * *base*, uint8_t *portAddr*, uint8_t *devAddr*, 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_MDIODC45Write\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

<i>base</i>	ENET peripheral base address.
<i>portAddr</i>	The MDIO port address(PHY address).
<i>devAddr</i>	The device address.
<i>data</i>	The data written to PHY.

8.8.18 static void ENET_StartExtC45SMIReadData (ENET_Type * *base*, uint8_t *portAddr*, uint8_t *devAddr*) [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_MDIODC45Read\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

<i>base</i>	ENET peripheral base address.
<i>portAddr</i>	The MDIO port address(PHY address).
<i>devAddr</i>	The device address.

8.8.19 status_t ENET_MDIODC45Write (ENET_Type * *base*, uint8_t *portAddr*, uint8_t *devAddr*, uint16_t *regAddr*, uint16_t *data*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>portAddr</i>	The MDIO port address(PHY address).
<i>devAddr</i>	The device address.
<i>regAddr</i>	The PHY register address.
<i>data</i>	The data written to PHY.

Returns

kStatus_Success MDIO access succeeds.
kStatus_Timeout MDIO access timeout.

8.8.20 status_t ENET_MDIODC45Read (ENET_Type * *base*, uint8_t *portAddr*, uint8_t *devAddr*, uint16_t *regAddr*, uint16_t * *pData*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>portAddr</i>	The MDIO port address(PHY address).
<i>devAddr</i>	The device address.
<i>regAddr</i>	The PHY register address.
<i>pData</i>	The data read from PHY.

Returns

kStatus_Success MDIO access succeeds.
kStatus_Timeout MDIO access timeout.

8.8.21 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.

8.8.22 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.

8.8.23 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.

8.8.24 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.

8.8.25 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.
-------------	-------------------------------

8.8.26 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.

8.8.27 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.

8.8.28 static void ENET_EnableInterrupts (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

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

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

Parameters

<i>base</i>	ENET peripheral base address.
<i>mask</i>	ENET interrupts to enable. This is a logical OR of the enumeration enet_interrupt_enable_t .

8.8.29 static void ENET_DisableInterrupts (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

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

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

Parameters

<i>base</i>	ENET peripheral base address.
<i>mask</i>	ENET interrupts to disable. This is a logical OR of the enumeration enet_interrupt_enable_t .

8.8.30 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](#).

8.8.31 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 <code>enet_interrupt_enable_t</code> .

8.8.32 void ENET_SetRxISRHandler (ENET_Type * *base*, enet_isr_ring_t *ISRHandler*)

Parameters

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

8.8.33 void ENET_SetTxISRHandler (ENET_Type * *base*, enet_isr_ring_t *ISRHandler*)

Parameters

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

8.8.34 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.

8.8.35 void ENET_GetRxErrBeforeReadFrame (enet_handle_t * *handle*, enet_data_error_stats_t * *eErrorStatic*, uint8_t *ringId*)

This API must be called after the ENET_GetRxFrameSize and before the [ENET_ReadFrame\(\)](#). If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```
*     status = ENET_GetRxFrameSize(&g_handle, &length, 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).

8.8.36 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.

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

8.8.38 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.

8.8.39 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_TxFrameBusy</i>	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with <i>kStatus_ENET_TxFrameBusy</i> .

8.8.40 **status_t ENET_SetTxReclaim (enet_handle_t * *handle*, bool *isEnable*, 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>isEnable</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.

8.8.41 **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.

8.8.42 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.

8.8.43 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.

8.8.44 void ENET_TransmitIRQHandler (ENET_Type * *base*, enet_handle_t * *handle*, uint32_t *ringId*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer.
<i>ringId</i>	The ring id or ring number.

8.8.45 void ENET_ReceiveIRQHandler (ENET_Type * *base*, enet_handle_t * *handle*, uint32_t *ringId*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer.
<i>ringId</i>	The ring id or ring number.

8.8.46 void ENET_CommonFrame1IRQHandler (ENET_Type * *base*)

This is used for the combined tx/rx interrupt for multi-ring (frame 1).

Parameters

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

8.8.47 void ENET_CommonFrame2IRQHandler (ENET_Type * *base*)

This is used for the combined tx/rx interrupt for multi-ring (frame 2).

Parameters

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

8.8.48 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.

8.8.49 void ENET_Ptp1588IRQHandler (ENET_Type * *base*)

This is used for the 1588 timer interrupt.

Parameters

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

8.8.50 void ENET_CommonFrame0IRQHandler (ENET_Type * *base*)

This is used for the combined tx/rx/error interrupt for single/mutli-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_PCG_TIME_SLOT_TOTAL_NUMBER` GPC_SLT_CFG_PU_COUNT
Total number of the timeslot.

Typedefs

- `typedef struct _gpc_lpm_config gpc_lpm_config_t`
configuration for enter DSM mode

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 _gpc_pgc_ack_sel.

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>powerDown-Slot</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_init_config](#)
Structure to configure the running mode. [More...](#)

Typedefs

- typedef enum [_gpt_clock_source](#) [gpt_clock_source_t](#)
List of clock sources.
- typedef enum [_gpt_input_capture_channel](#) [gpt_input_capture_channel_t](#)
List of input capture channel number.
- typedef enum [_gpt_input_operation_mode](#) [gpt_input_operation_mode_t](#)
List of input capture operation mode.

- `typedef enum _gpt_output_compare_channel gpt_output_compare_channel_t`
List of output compare channel number.
- `typedef enum _gpt_output_operation_mode gpt_output_operation_mode_t`
List of output compare operation mode.
- `typedef enum _gpt_interrupt_enable gpt_interrupt_enable_t`
List of GPT interrupts.
- `typedef enum _gpt_status_flag gpt_status_flag_t`
Status flag.
- `typedef struct _gpt_init_config gpt_config_t`
Structure to configure the running mode.

Enumerations

- `enum _gpt_clock_source {`
`kGPT_ClockSource_Off = 0U,`
`kGPT_ClockSource_Pерiph = 1U,`
`kGPT_ClockSource_HighFreq = 2U,`
`kGPT_ClockSource_Ext = 3U,`
`kGPT_ClockSource_LowFreq = 4U,`
`kGPT_ClockSource_Osc = 5U }`
List of clock sources.
- `enum _gpt_input_capture_channel {`
`kGPT_InputCapture_Channel1 = 0U,`
`kGPT_InputCapture_Channel2 = 1U }`
List of input capture channel number.
- `enum _gpt_input_operation_mode {`
`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 {`
`kGPT_OutputCompare_Channel1 = 0U,`
`kGPT_OutputCompare_Channel2 = 1U,`
`kGPT_OutputCompare_Channel3 = 2U }`
List of output compare channel number.
- `enum _gpt_output_operation_mode {`
`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 {`

```
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` {


```
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)

- 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_init_config

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_init_config::clockSource](#)
- (2) [uint32_t _gpt_init_config::divider](#)
- (3) [bool _gpt_init_config::enableFreeRun](#)
- (4) [bool _gpt_init_config::enableRunInWait](#)
- (5) [bool _gpt_init_config::enableRunInStop](#)
- (6) [bool _gpt_init_config::enableRunInDoze](#)
- (7) [bool _gpt_init_config::enableRunInDbg](#)
- (8) [bool _gpt_init_config::enableMode](#)

10.5 Typedef Documentation

10.5.1 [typedef enum _gpt_clock_source gpt_clock_source_t](#)

Note

Actual number of clock sources is SoC dependent

10.5.2 `typedef enum _gpt_input_capture_channel gpt_input_capture_channel_t`

10.5.3 `typedef enum _gpt_input_operation_mode gpt_input_operation_mode_t`

10.5.4 `typedef enum _gpt_output_compare_channel gpt_output_compare_channel_t`

10.5.5 `typedef enum _gpt_output_operation_mode gpt_output_operation_mode_t`

10.5.6 `typedef enum _gpt_status_flag gpt_status_flag_t`

10.5.7 `typedef struct _gpt_init_config gpt_config_t`

10.6 Enumeration Type Documentation

10.6.1 `enum _gpt_clock_source`

Note

Actual number of clock sources is SoC dependent

Enumerator

kGPT_ClockSource_Off GPT Clock Source Off.

kGPT_ClockSource_Pерiph GPT Clock Source from Peripheral Clock.

kGPT_ClockSource_HighFreq GPT Clock Source from High Frequency Reference Clock.

kGPT_ClockSource_Ext GPT Clock Source from external pin.

kGPT_ClockSource_LowFreq GPT Clock Source from Low Frequency Reference Clock.

kGPT_ClockSource_Osc GPT Clock Source from Crystal oscillator.

10.6.2 `enum _gpt_input_capture_channel`

Enumerator

kGPT_InputCapture_Channel1 GPT Input Capture Channel1.

kGPT_InputCapture_Channel2 GPT Input Capture Channel2.

10.6.3 enum _gpt_input_operation_mode

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.

10.6.4 enum _gpt_output_compare_channel

Enumerator

kGPT_OutputCompare_Channel1 Output Compare Channel1.

kGPT_OutputCompare_Channel2 Output Compare Channel2.

kGPT_OutputCompare_Channel3 Output Compare Channel3.

10.6.5 enum _gpt_output_operation_mode

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.

10.6.6 enum _gpt_interrupt_enable

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 Channell1 interrupt enable.

kGPT_RollOverFlagInterruptEnable Counter rolled over interrupt enable.

10.6.7 enum _gpt_status_flag

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.

10.7 Function Documentation

10.7.1 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.

10.7.2 void GPT_Deinit (*GPT_Type* * *base*)

Parameters

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

10.7.3 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.
---------------	--

10.7.4 static void GPT_SoftwareReset (**GPT_Type** * *base*) [inline], [static]

Parameters

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

10.7.5 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 gpt_clock_source_t typedef enumeration).

10.7.6 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).

10.7.7 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).

10.7.8 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).

10.7.9 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).

10.7.10 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).

10.7.11 static void GPT_StartTimer (**GPT_Type** * *base*) [inline], [static]

Parameters

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

10.7.12 static void GPT_StopTimer (**GPT_Type** * *base*) [inline], [static]

Parameters

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

10.7.13 static uint32_t GPT_GetCurrentTimerCount (**GPT_Type** * *base*) [inline], [static]

Parameters

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

Returns

Current GPT counter value.

10.7.14 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 gpt_input_capture_channel_t typedef enumeration).
<i>mode</i>	GPT input capture operation mode (see gpt_input_operation_mode_t typedef enumeration).

10.7.15 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 gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture operation mode (see [gpt_input_operation_mode_t](#) typedef enumeration).

10.7.16 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 gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

10.7.17 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 gpt_output_compare_channel_t typedef enumeration).
<i>mode</i>	GPT output operation mode (see gpt_output_operation_mode_t typedef enumeration).

10.7.18 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 gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output operation mode (see [gpt_output_operation_mode_t](#) typedef enumeration).

10.7.19 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 gpt_output_compare_channel_t typedef enumeration).
<i>value</i>	GPT output compare value.

10.7.20 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 gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output compare value.

10.7.21 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 gpt_output_compare_channel_t typedef enumeration).

10.7.22 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 gpt_interrupt_enable_t

10.7.23 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 gpt_interrupt_enable_t

10.7.24 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](#)

10.7.25 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 gpt_status_flag_t for bit definition).

Returns

GPT status, each bit represents one status flag.

10.7.26 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 gpt_status_flag_t 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](#)
GPIO Init structure definition. [More...](#)

Typedefs

- typedef enum [_gpio_pin_direction](#) [gpio_pin_direction_t](#)
GPIO direction definition.
- typedef enum [_gpio_interrupt_mode](#) [gpio_interrupt_mode_t](#)
GPIO interrupt mode definition.
- typedef struct [_gpio_pin_config](#) [gpio_pin_config_t](#)
GPIO Init structure definition.

Enumerations

- enum [_gpio_pin_direction](#) {
 [kGPIO_DigitalInput](#) = 0U,
 [kGPIO_DigitalOutput](#) = 1U }
GPIO direction definition.
- enum [_gpio_interrupt_mode](#) {
 [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)

- 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

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::direction](#)
- (2) [gpio_interrupt_mode_t _gpio_pin_config::interruptMode](#)

11.4 Macro Definition Documentation

11.4.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))

11.5 Typedef Documentation

11.5.1 [typedef enum _gpio_pin_direction gpio_pin_direction_t](#)

11.5.2 [typedef enum _gpio_interrupt_mode gpio_interrupt_mode_t](#)

11.5.3 [typedef struct _gpio_pin_config gpio_pin_config_t](#)

11.6 Enumeration Type Documentation

11.6.1 enum _gpio_pin_direction

Enumerator

kGPIO_DigitalInput Set current pin as digital input.

kGPIO_DigitalOutput Set current pin as digital output.

11.6.2 enum _gpio_interrupt_mode

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.

11.7 Function Documentation

11.7.1 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 gpio_pin_config_t structure that contains the configuration information.

11.7.2 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. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

11.7.3 static void GPIO_WritePinOutput (**GPIO_Type** * *base*, **uint32_t** *pin*, **uint8_t** *output*) [inline], [static]

Deprecated Do not use this function. It has been superceded by [GPIO_PinWrite](#).

11.7.4 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

11.7.5 static void GPIO_SetPinsOutput (**GPIO_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by [GPIO_PortSet](#).

11.7.6 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

**11.7.7 static void GPIO_Clear PinsOutput (GPIO_Type * *base*, uint32_t *mask*)
[inline], [static]**

Deprecated Do not use this function. It has been superceded by [GPIO_PortClear](#).

**11.7.8 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

**11.7.9 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.
-------------	-------------------

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

Deprecated Do not use this function. It has been superceded by [GPIO_PinRead](#).

11.7.11 **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.
-------------	-----------------------

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

Deprecated Do not use this function. It has been superceded by [GPIO_PinReadPadStatus](#).

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

Parameters

<i>base</i>	GPIO base pointer.
<i>pin</i>	GPIO port pin number.
<i>pinInterrupt-Mode</i>	pointer to a gpio_interrupt_mode_t structure that contains the interrupt mode information.

11.7.14 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 superceded by [GPIO_PinSetInterruptConfig](#).

11.7.15 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.

**11.7.16 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.

**11.7.17 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.

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

Deprecated Do not use this function. It has been superceded by [GPIO_PortDisableInterrupts](#).

**11.7.19 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.
----------------	----------------------------

11.7.20 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.
----------------	----------------------------

11.7.21 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.

11.7.22 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](#)

- struct [_i2c_master_transfer](#)
I2C master transfer structure. [More...](#)
- struct [_i2c_master_handle](#)
I2C master handle structure. [More...](#)
- struct [_i2c_slave_config](#)
I2C slave user configuration. [More...](#)
- struct [_i2c_slave_transfer](#)
I2C slave transfer structure. [More...](#)
- struct [_i2c_slave_handle](#)
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 enum [_i2c_direction](#) [i2c_direction_t](#)
The direction of master and slave transfers.
- typedef struct [_i2c_master_config](#) [i2c_master_config_t](#)
I2C master user configuration.
- typedef struct [_i2c_master_handle](#) [i2c_master_handle_t](#)
I2C master handle typedef.
- 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 struct [_i2c_master_transfer](#) [i2c_master_transfer_t](#)
I2C master transfer structure.
- typedef enum [_i2c_slave_transfer_event](#) [i2c_slave_transfer_event_t](#)
Set of events sent to the callback for nonblocking slave transfers.
- typedef struct [_i2c_slave_handle](#) [i2c_slave_handle_t](#)
I2C slave handle typedef.
- typedef struct [_i2c_slave_config](#) [i2c_slave_config_t](#)
I2C slave user configuration.
- typedef struct [_i2c_slave_transfer](#) [i2c_slave_transfer_t](#)
I2C slave transfer structure.
- 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_IIEN_MASK }

I2C feature interrupt source.

- enum _i2c_direction {

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 {

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 srcClock_Hz)
Initializes the I2C peripheral.
- void **I2C_MasterDeinit** (I2C_Type *base)
De-initializes the I2C master peripheral.
- void **I2C_MasterGetDefaultConfig** (**i2c_master_config_t** *masterConfig)
Sets the I2C master configuration structure to default values.
- void **I2C_SlaveInit** (I2C_Type *base, const **i2c_slave_config_t** *slaveConfig)
Initializes the I2C peripheral.
- void **I2C_SlaveDeinit** (I2C_Type *base)
De-initializes the I2C slave peripheral.
- void **I2C_SlaveGetDefaultConfig** (**i2c_slave_config_t** *slaveConfig)
Sets the I2C slave configuration structure to default values.
- static void **I2C_Enable** (I2C_Type *base, bool enable)
Enables or disables the I2C peripheral operation.

Status

- static uint32_t **I2C_MasterGetStatusFlags** (I2C_Type *base)
Gets the I2C status flags.
- static void **I2C_MasterClearStatusFlags** (I2C_Type *base, uint32_t statusMask)
Clears the I2C status flag state.
- static uint32_t **I2C_SlaveGetStatusFlags** (I2C_Type *base)
Gets the I2C status flags.
- static void **I2C_SlaveClearStatusFlags** (I2C_Type *base, uint32_t statusMask)
Clears the I2C status flag state.

Interrupts

- void **I2C_EnableInterrupts** (I2C_Type *base, uint32_t mask)
Enables I2C interrupt requests.
- void **I2C_DisableInterrupts** (I2C_Type *base, uint32_t mask)
Disables I2C interrupt requests.

Bus Operations

- void **I2C_MasterSetBaudRate** (I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
Sets the I2C master transfer baud rate.
- **status_t I2C_MasterStart** (I2C_Type *base, uint8_t address, **i2c_direction_t** direction)
Sends a START on the I2C bus.
- **status_t I2C_MasterStop** (I2C_Type *base)
Sends a STOP signal on the I2C bus.
- **status_t I2C_MasterRepeatedStart** (I2C_Type *base, uint8_t address, **i2c_direction_t** direction)
Sends a REPEATED START on the I2C bus.

- `status_t I2C_MasterWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize, uint32_t flags)`
Performs a polling send transaction on the I2C bus.
- `status_t I2C_MasterReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize, uint32_t flags)`
Performs a polling receive transaction on the I2C bus.
- `status_t I2C_SlaveWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize)`
Performs a polling send transaction on the I2C bus.
- `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

Data Fields

- `bool enableMaster`

- **uint32_t baudRate_Bps**
Baud rate configuration of I2C peripheral.

Field Documentation

- (1) **bool _i2c_master_config::enableMaster**
- (2) **uint32_t _i2c_master_config::baudRate_Bps**

12.2.3.2 struct _i2c_master_transfer

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::flags**
- (2) **uint8_t _i2c_master_transfer::slaveAddress**
- (3) **i2c_direction_t _i2c_master_transfer::direction**
- (4) **uint32_t _i2c_master_transfer::subaddress**

Transferred MSB first.

- (5) `uint8_t _i2c_master_transfer::subaddressSize`
- (6) `uint8_t* volatile _i2c_master_transfer::data`
- (7) `volatile size_t _i2c_master_transfer::dataSize`

12.2.3.3 struct _i2c_master_handle

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::transfer`
- (2) `size_t _i2c_master_handle::transferSize`
- (3) `uint8_t _i2c_master_handle::state`
- (4) `i2c_master_transfer_callback_t _i2c_master_handle::completionCallback`
- (5) `void* _i2c_master_handle::userData`

12.2.3.4 struct _i2c_slave_config

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::enableSlave**
- (2) **uint16_t _i2c_slave_config::slaveAddress**

12.2.3.5 struct _i2c_slave_transfer

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::event**
- (2) **uint8_t* volatile _i2c_slave_transfer::data**
- (3) **volatile size_t _i2c_slave_transfer::dataSize**
- (4) **status_t _i2c_slave_transfer::completionStatus**

Only applies for [kI2C_SlaveCompletionEvent](#).

- (5) **size_t _i2c_slave_transfer::transferredCount**

12.2.3.6 struct _i2c_slave_handle

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::state
- (2) i2c_slave_transfer_t _i2c_slave_handle::transfer
- (3) uint32_t _i2c_slave_handle::eventMask
- (4) i2c_slave_transfer_callback_t _i2c_slave_handle::callback
- (5) void* _i2c_slave_handle::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 enum _i2c_direction i2c_direction_t

12.2.5.2 typedef struct _i2c_master_config i2c_master_config_t

12.2.5.3 typedef struct _i2c_master_handle i2c_master_handle_t

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

12.2.5.5 typedef struct _i2c_master_transfer i2c_master_transfer_t

12.2.5.6 typedef enum _i2c_slave_transfer_event 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.

12.2.5.7 `typedef struct _i2c_slave_handle i2c_slave_handle_t`**12.2.5.8 `typedef struct _i2c_slave_config i2c_slave_config_t`****12.2.5.9 `typedef struct _i2c_slave_transfer i2c_slave_transfer_t`****12.2.5.10 `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

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

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"> • kI2C_ArbitrationLostFlag • kI2C_IntPendingFlag

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>	<p>The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:</p> <ul style="list-style-type: none"> • kI2C_IntPendingFlagFlag

12.2.7.12 void I2C_EnableInterrupts (I2C_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	I2C base pointer
<i>mask</i>	<p>interrupt source The parameter can be combination of the following source if defined:</p> <ul style="list-style-type: none"> • kI2C_GlobalInterruptEnable • kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable • kI2C_SdaTimeoutInterruptEnable

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">• kI2C_GlobalInterruptEnable• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable• kI2C_SdaTimeoutInterruptEnable

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 <i>kI2C_TransferDefaultFlag</i> to issue a stop and <i>kI2C_TransferNoStop</i> to not send a stop.

Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_ArbitrationLost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.

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_ArbitrationLost</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 i2c_master_transfer_t 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 i2c_slave_handle_t structure to store the transfer state.
<i>callback</i>	pointer to user callback function.
<i>userData</i>	user parameter passed to the callback function.

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 <code>i2c_slave_transfer_event_t</code> 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 <code>kI2C_SlaveAllEvents</code> to enable all events.

Return values

<code>kStatus_Success</code>	Slave transfers were successfully started.
<code>kStatus_I2C_Busy</code>	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 <code>i2c_slave_handle_t</code> 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 <code>i2c_slave_handle_t</code> 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

Typedefs

- `typedef enum _pwm_clock_source pwm_clock_source_t`
PWM clock source select.
- `typedef enum _pwm_fifo_water_mark pwm_fifo_water_mark_t`
PWM FIFO water mark select.
- `typedef enum _pwm_byte_data_swap pwm_byte_data_swap_t`
PWM byte data swap select.
- `typedef enum _pwm_half_word_data_swap pwm_half_word_data_swap_t`
PWM half-word data swap select.
- `typedef enum _pwm_output_configuration pwm_output_configuration_t`
PWM Output Configuration.
- `typedef enum _pwm_sample_repeat pwm_sample_repeat_t`
PWM FIFO sample repeat It determines the number of times each sample from the FIFO is to be used.
- `typedef enum _pwm_interrupt_enable pwm_interrupt_enable_t`
List of PWM interrupt options.
- `typedef enum _pwm_status_flags pwm_status_flags_t`

- **typedef enum _pwm_fifo_available pwm_fifo_available_t**
List of PWM FIFO available.

Enumerations

- **enum _pwm_clock_source {**
kPWM_PeripheralClock = 1U,
kPWM_HighFrequencyClock,
kPWM_LowFrequencyClock }
PWM clock source select.
- **enum _pwm_fifo_water_mark {**
kPWM_FIFOWaterMark_1 = 0U,
kPWM_FIFOWaterMark_2,
kPWM_FIFOWaterMark_3,
kPWM_FIFOWaterMark_4 }
PWM FIFO water mark select.
- **enum _pwm_byte_data_swap {**
kPWM_BytNoSwap = 0U,
kPWM_ByteSwap }
PWM byte data swap select.
- **enum _pwm_half_word_data_swap {**
kPWM_HalfWordNoSwap = 0U,
kPWM_HalfWordSwap }
PWM half-word data swap select.
- **enum _pwm_output_configuration {**
kPWM_SetAtRolloverAndClearAtcomparison = 0U,
kPWM_ClearAtRolloverAndSetAtcomparison,
kPWM_NoConfigure }
PWM Output Configuration.
- **enum _pwm_sample_repeat {**
kPWM_EachSampleOnce = 0u,
kPWM_EachSampltwice,
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 {**
kPWM_FIFOEmptyInterruptEnable = (1U << 0),
kPWM_RolloverInterruptEnable = (1U << 1),
kPWM_CompareInterruptEnable = (1U << 2) }
List of PWM interrupt options.
- **enum _pwm_status_flags {**
kPWM_FIFOEmptyFlag = (1U << 3),
kPWM_RolloverFlag = (1U << 4),
kPWM_CompareFlag = (1U << 5),
kPWM_FIFOWriteErrorFlag }
List of PWM status flags.

- enum _pwm_fifo_available {
 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 Typedef Documentation

13.4.1 `typedef enum _pwm_clock_source pwm_clock_source_t`

13.4.2 `typedef enum _pwm_fifo_water_mark pwm_fifo_water_mark_t`

Sets the data level at which the FIFO empty flag will be set

13.4.3 `typedef enum _pwm_byte_data_swap 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.

13.4.4 `typedef enum _pwm_half_word_data_swap pwm_half_word_data_swap_t`

13.5 Enumeration Type Documentation

13.5.1 `enum _pwm_clock_source`

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.

13.5.2 `enum _pwm_fifo_water_mark`

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.

13.5.3 enum _pwm_byte_data_swap

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

13.5.4 enum _pwm_half_word_data_swap

Enumerator

kPWM_HalfWordNoSwap Half word swapping does not take place.

kPWM_HalfWordSwap Half word from write data bus are swapped.

13.5.5 enum _pwm_output_configuration

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.

13.5.6 enum _pwm_sample_repeat

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.

13.5.7 enum _pwm_interrupt_enable

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.

13.5.8 enum _pwm_status_flags

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.

13.5.9 enum _pwm_fifo_available

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

13.6 Function Documentation

13.6.1 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.

13.6.2 void PWM_Deinit (PWM_Type * *base*)

Parameters

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

13.6.3 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.
---------------	---

13.6.4 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
-------------	-----------------------------

13.6.5 static void PWM_StopTimer (**PWM_Type** * *base*) [inline], [static]

Parameters

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

13.6.6 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
-------------	-----------------------------

13.6.7 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 pwm_interrupt_enable_t

13.6.8 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 pwm_interrupt_enable_t

13.6.9 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](#)

13.6.10 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](#)

13.6.11 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 pwm_status_flags_t

13.6.12 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](#)

13.6.13 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.

13.6.14 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.

13.6.15 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 (Hz) = PCLK(Hz) / (period +2)

13.6.16 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.

13.6.17 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_RxRingBufferOverrun`. In the callback function, the upper layer reads data out from the ring buffer. If not, existing data is overwritten by the new data.

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

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`
UART configuration structure. [More...](#)
- struct `_uart_transfer`
UART transfer structure. [More...](#)
- struct `_uart_handle`
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 enum `_uart_data_bits` `uart_data_bits_t`
UART data bits count.
- typedef enum `_uart_parity_mode` `uart_parity_mode_t`
UART parity mode.
- typedef enum `_uart_stop_bit_count` `uart_stop_bit_count_t`
UART stop bit count.
- typedef enum `_uart_idle_condition` `uart_idle_condition_t`
UART idle condition detect.
- typedef struct `_uart_config` `uart_config_t`
UART configuration structure.
- typedef struct `_uart_transfer` `uart_transfer_t`
UART transfer structure.
- typedef struct `_uart_handle` `uart_handle_t`
Forward declaration of the handle typedef.

- `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_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_IUART, 8),`
 `kStatus_UART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_IUART, 9),`
 `kStatus_UART_NoiseError = MAKE_STATUS(kStatusGroup_IUART, 10),`
 `kStatus_UART_FramingError = MAKE_STATUS(kStatusGroup_IUART, 11),`
 `kStatus_UART_ParityError = MAKE_STATUS(kStatusGroup_IUART, 12),`
 `kStatus_UART_BaudrateNotSupport,`
 `kStatus_UART_BreakDetect = MAKE_STATUS(kStatusGroup_IUART, 14),`
 `kStatus_UART_Timeout = MAKE_STATUS(kStatusGroup_IUART, 15) }`

Error codes for the UART driver.

- `enum _uart_data_bits {`
 `kUART_SevenDataBits = 0x0U,`
 `kUART_EightDataBits = 0x1U }`
UART data bits count.
- `enum _uart_parity_mode {`
 `kUART_ParityDisabled = 0x0U,`
 `kUART_ParityEven = 0x2U,`
 `kUART_ParityOdd = 0x3U }`
UART parity mode.
- `enum _uart_stop_bit_count {`
 `kUART_OneStopBit = 0x0U,`
 `kUART_TwoStopBit = 0x1U }`
UART stop bit count.
- `enum _uart_idle_condition {`
 `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_tAWakeFlag = 0x00940005U,
kUART_AwakeFlag = 0x00940004U,
kUART_Rs485SlaveAddrMatchFlag = 0x00940003U,
kUART_AutoBaudFlag = 0x0098000FU,
kUART_TxEmptyFlag = 0x0098000EU,
kUART_DtrFlag = 0x0098000DU,
kUART_IdleFlag = 0x0098000CU,
kUART_AutoBaudCntStopFlag = 0x0098000BU,
kUART_RiDeltaFlag = 0x0098000AU,
kUART_RiFlag = 0x00980009U,
kUART_IrFlag = 0x00980008U,
kUART_WakeFlag = 0x00980007U,
kUART_DcdDeltaFlag = 0x00980006U,
kUART_DcdFlag = 0x00980005U,
kUART_RtsFlag = 0x00980004U,
kUART_TxCompleteFlag = 0x00980003U,
kUART_BreakDetectFlag = 0x00980002U,
kUART_RxOverrunFlag = 0x00980001U,
kUART_RxDataReadyFlag = 0x00980000U }

```

UART status flags.

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)

- `uint32_t UART_GetEnabledInterrupts (UART_Type *base)`
Disables the UART interrupts according to the provided mask.

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_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

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::baudRate_Bps`
- (2) `uart_parity_mode_t _uart_config::parityMode`
- (3) `uart_stop_bit_count_t _uart_config::stopBitCount`

14.2.3.2 struct _uart_transfer**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::data`
- (2) `uint8_t* _uart_transfer::rxData`
- (3) `const uint8_t* _uart_transfer::txData`
- (4) `size_t _uart_transfer::dataSize`

14.2.3.3 struct _uart_handle**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`

- `uint8_t *volatile rxData`
Size of the data to send out.
- `volatile size_t rxDataSize`
Address of remaining data to receive.
- `size_t rxDataSizeAll`
Size of the remaining data to receive.
- `uint8_t * rxRingBuffer`
Size of the data to receive.
- `size_t rxRingBufferSize`
Start address of the receiver ring buffer.
- `volatile uint16_t rxRingBufferHead`
Size of the ring buffer.
- `volatile uint16_t rxRingBufferTail`
Index for the driver to store received data into ring buffer.
- `uart_transfer_callback_t callback`
Index for the user to get data from the ring buffer.
- `void * userData`
Callback function.
- `volatile uint8_t txState`
UART callback function parameter.
- `volatile uint8_t rxState`
TX transfer state.
- `volatile uint8_t rxState`
RX transfer state.

Field Documentation

- (1) `const uint8_t* volatile _uart_handle::txData`
- (2) `volatile size_t _uart_handle::txDataSize`
- (3) `size_t _uart_handle::txDataSizeAll`
- (4) `uint8_t* volatile _uart_handle::rxData`
- (5) `volatile size_t _uart_handle::rxDataSize`
- (6) `size_t _uart_handle::rxDataSizeAll`
- (7) `uint8_t* _uart_handle::rxRingBuffer`
- (8) `size_t _uart_handle::rxRingBufferSize`
- (9) `volatile uint16_t _uart_handle::rxRingBufferHead`
- (10) `volatile uint16_t _uart_handle::rxRingBufferTail`
- (11) `uart_transfer_callback_t _uart_handle::callback`
- (12) `void* _uart_handle::userData`
- (13) `volatile uint8_t _uart_handle::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 enum _uart_data_bits uart_data_bits_t

14.2.5.2 typedef enum _uart_parity_mode uart_parity_mode_t

14.2.5.3 typedef enum _uart_stop_bit_count uart_stop_bit_count_t

14.2.5.4 typedef enum _uart_idle_condition uart_idle_condition_t

14.2.5.5 typedef struct _uart_config uart_config_t

14.2.5.6 typedef struct _uart_transfer uart_transfer_t

14.2.5.7 typedef struct _uart_handle uart_handle_t

14.2.5.8 typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t

kStatus_UART_RxBusy Receiver is busy.
kStatus_UART_TxIdle UART transmitter is idle.
kStatus_UART_RxIdle UART receiver is idle.
kStatus_UART_TxWatermarkTooLarge TX FIFO watermark too large.
kStatus_UART_RxWatermarkTooLarge RX FIFO watermark too large.
kStatus_UART_FlagCannotClearManually UART flag can't be manually cleared.
kStatus_UART_Error Error happens on UART.
kStatus_UART_RxRingBufferOverrun UART RX software ring buffer overrun.
kStatus_UART_RxHardwareOverrun UART RX receiver overrun.
kStatus_UART_NoiseError UART noise error.
kStatus_UART_FramingError UART framing error.
kStatus_UART_ParityError UART parity error.
kStatus_UART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_UART_BreakDetect Receiver detect BREAK signal.
kStatus_UART_Timeout UART times out.

14.2.6.2 enum _uart_data_bits

Enumerator

kUART_SevenDataBits Seven data bit.
kUART_EightDataBits Eight data bit.

14.2.6.3 enum _uart_parity_mode

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

Enumerator

kUART_OneStopBit One stop bit.
kUART_TwoStopBit Two stop bits.

14.2.6.5 enum _uart_idle_condition

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_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the UART.

```
* uart_config_t uartConfig;
* uartConfig.baudRate_Bps = 115200U;
* uartConfig.parityMode = kUART_ParityDisabled;
* uartConfig.dataBitsCount = kUART_EightDataBits;
* uartConfig.stopBitCount = kUART_OneStopBit;
* uartConfig.txFifoWatermark = 2;
* uartConfig.rxFifoWatermark = 1;
* uartConfig.enableAutoBaudrate = false;
* uartConfig.enableTx = true;
* uartConfig.enableRx = true;
* UART\_Init(UART1, &uartConfig, 24000000U);
*
```

Parameters

<i>base</i>	UART peripheral base address.
<i>config</i>	Pointer to a user-defined configuration structure.
<i>srcClock_Hz</i>	UART clock source frequency in HZ.

Return values

<i>kStatus_Success</i>	UART initialize succeed
------------------------	-------------------------

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 = flase; 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, 20000000U);
*
```

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 uart_idle_condition_t .

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 _uart_interrupt_enable .

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 _uart_interrupt_enable .

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_TxTransmissionCompleteFlag](#) to ensure that the TX is finished.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure. See uart_transfer_t .

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_NoTransferIn-Progress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	The parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

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 [k_Status_UART_RxIdle](#). For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the *xfer->data* and this function returns with the parameter *receivedBytes* set to 5. For the left 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the UART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the *xfer->data*. When all data is received, the upper layer is notified.

Parameters

<i>base</i>	UART peripheral base address.
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART transfer structure, see uart_transfer_t .
<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_NoTransferIn-Progress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <i>count</i> ;

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 Tx FIFO 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 Rx FIFO 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 Rx FIFO 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"> • true: Enable Automatic Baud Rate Detection feature. • false: Disable Automatic Baud Rate Detection feature.

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`
UART configuration structure. [More...](#)

Typedefs

- typedef struct `_uart_rtos_config` `uart_rtos_config_t`
UART configuration structure.

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`

Data Fields

- `UART_Type * base`
UART base address.

- `uint32_t srclk`
UART source clock in Hz.
- `uint32_t baudrate`
Desired communication speed.
- `uart_parity_mode_t parity`
Parity setting.
- `uart_stop_bit_count_t stopbits`
Number of stop bits to use.
- `uint8_t * buffer`
Buffer for background reception.
- `uint32_t buffer_size`
Size of buffer for background reception.

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

<code>handle</code>	The RTOS UART handle, the pointer to an allocated space for RTOS context.
<code>t_handle</code>	The pointer to the allocated space to store the transactional layer internal state.
<code>cfg</code>	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 <i>size_t</i> where the number of received data is filled.

14.4 UART SDMA Driver

14.4.1 Overview

Data Structures

- struct `_uart_sdma_handle`
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::callback`
- (2) `void* _uart_sdma_handle::userData`
- (3) `size_t _uart_sdma_handle::rxDataSizeAll`
- (4) `size_t _uart_sdma_handle::txDataSizeAll`
- (5) `sdma_handle_t* _uart_sdma_handle::txSdmaHandle`
- (6) `sdma_handle_t* _uart_sdma_handle::rxSdmaHandle`
- (7) `volatile uint8_t _uart_sdma_handle::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 uart_transfer_t .

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 uart_transfer_t .

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 uninitialized 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.

Typedefs

- **typedef enum _mu_msg_reg_index mu_msg_reg_index_t**
MU message register.

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,
 kMU_OtherSideInResetFlag = MU_SR_RS_MASK }
- *MU status flags.*
- **enum _mu_interrupt_enable {**
 kMU_Tx0EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 3U)),
 kMU_Tx1EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 2U)),
 kMU_Tx2EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 1U)),
 kMU_Tx3EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 0U)),
 kMU_Rx0FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 3U)),
 kMU_Rx1FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 2U)),
 kMU_Rx2FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 1U)),
 kMU_Rx3FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 0U)),
 kMU_GenInt0InterruptEnable = (int)(1U << (MU_CR_GIEn_SHIFT + 3U)),
 kMU_GenInt1InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 2U)),
 kMU_GenInt2InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 1U)),
 kMU_GenInt3InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 0U)) }
- *MU interrupt source to enable.*
- **enum _mu_interrupt_trigger {**
 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**
MU message register.

Driver version

- #define **FSL_MU_DRIVER_VERSION** (MAKE_VERSION(2, 1, 3))
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.

MU misc functions

- static void **MU_MaskHardwareReset** (MU_Type *base, bool mask)
Mask hardware reset by 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, 3))

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.
kMU_OtherSideInResetFlag The other side is in reset.

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

<i>kMU_GenInt0InterruptTrigger</i>	General purpose interrupt 0.
<i>kMU_GenInt1InterruptTrigger</i>	General purpose interrupt 1.
<i>kMU_GenInt2InterruptTrigger</i>	General purpose interrupt 2.
<i>kMU_GenInt3InterruptTrigger</i>	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 mu_msg_reg_index_t .
<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 mu_msg_reg_index_t .
<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>regIndex</i>	RX register index, see mu_msg_reg_index_t .

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 mu_msg_reg_index_t

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>	<p>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.</p> <ul style="list-style-type: none"> • kMU_Tx0EmptyFlag • kMU_Tx1EmptyFlag • kMU_Tx2EmptyFlag • kMU_Tx3EmptyFlag • kMU_Rx0FullFlag • kMU_Rx1FullFlag • kMU_Rx2FullFlag • kMU_Rx3FullFlag • kMU_EventPendingFlag • kMU_FlagsUpdatingFlag • kMU_OtherSideInResetFlag

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_MaskHardwareReset (MU_Type * *base*, bool *mask*) [inline], [static]

The other core could call `MU_HardwareResetOtherCore()` to reset current core. To mask the reset, call this function and pass in true.

Parameters

<i>base</i>	MU peripheral base address.
<i>mask</i>	Pass true to mask the hardware reset, pass false to unmask it.

15.5.17 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_TransferReceive-NonBlocking\(\)](#) 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](#)
PDM channel configurations. [More...](#)
- struct [_pdm_config](#)
PDM user configuration structure. [More...](#)
- struct [_pdm_hwvad_config](#)
PDM voice activity detector user configuration structure. [More...](#)
- struct [_pdm_hwvad_noise_filter](#)
PDM voice activity detector noise filter user configuration structure. [More...](#)
- struct [_pdm_hwvad_zero_cross_detector](#)
PDM voice activity detector zero cross detector configuration structure. [More...](#)
- struct [_pdm_transfer](#)
PDM SDMA transfer structure. [More...](#)
- struct [_pdm_hwvad_notification](#)
PDM HWVAD notification structure. [More...](#)
- struct [_pdm_handle](#)
PDM handle structure. [More...](#)

Macros

- #define [PDM_XFER_QUEUE_SIZE](#) (4U)
PDM XFER QUEUE SIZE.

Typedefs

- typedef enum [_pdm_dc_remover](#) pdm_dc_remover_t
PDM DC remover configurations.
- typedef enum [_pdm_df_quality_mode](#) pdm_df_quality_mode_t
PDM decimation filter quality mode.
- typedef enum [_pdm_df_output_gain](#) pdm_df_output_gain_t
PDM decimation filter output gain.
- typedef struct [_pdm_channel_config](#) pdm_channel_config_t
PDM channel configurations.
- typedef struct [_pdm_config](#) pdm_config_t
PDM user configuration structure.
- typedef enum [_pdm_hwvad_hpf_config](#) pdm_hwvad_hpf_config_t
High pass filter configure cut-off frequency.
- typedef enum [_pdm_hwvad_filter_status](#) pdm_hwvad_filter_status_t
HWVAD internal filter status.
- typedef struct [_pdm_hwvad_config](#) pdm_hwvad_config_t

- *PDM voice activity detector user configuration structure.*
- **typedef struct**
`_pdm_hwvad_noise_filter pdm_hwvad_noise_filter_t`
PDM voice activity detector noise filter user configuration structure.
- **typedef enum** `_pdm_hwvad_zcd_result pdm_hwvad_zcd_result_t`
PDM voice activity detector zero cross detector result.
- **typedef struct**
`_pdm_hwvad_zero_cross_detector pdm_hwvad_zero_cross_detector_t`
PDM voice activity detector zero cross detector configuration structure.
- **typedef struct** `_pdm_transfer pdm_transfer_t`
PDM SDMA transfer structure.
- **typedef struct** `_pdm_handle pdm_handle_t`
PDM handle.
- **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.
- **typedef struct**
`_pdm_hwvad_notification pdm_hwvad_notification_t`
PDM HWVAD notification structure.

Enumerations

- **enum {**
`kStatus_PDM_Busy = MAKE_STATUS(kStatusGroup_PDM, 0),`
`kStatus_PDM_CLK_LOW = MAKE_STATUS(kStatusGroup_PDM, 1),`
`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_StatusFrequencyLow = PDM_STAT_LOWFREQF_MASK,
kPDM_StatusCh0FifoDataAvailable = PDM_STAT_CH0F_MASK,
kPDM_StatusCh1FifoDataAvailable = PDM_STAT_CH1F_MASK,
kPDM_StatusCh2FifoDataAvailable = PDM_STAT_CH2F_MASK,
kPDM_StatusCh3FifoDataAvailable = PDM_STAT_CH3F_MASK,
kPDM_StatusCh4FifoDataAvailable = PDM_STAT_CH4F_MASK,
kPDM_StatusCh5FifoDataAvailable = PDM_STAT_CH5F_MASK,
kPDM_StatusCh6FifoDataAvailable = PDM_STAT_CH6F_MASK,
kPDM_StatusCh7FifoDataAvailable = 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_FIFOUND7_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_remover` {

kPDM_DcRemoverCutOff21Hz = 0U,
 kPDM_DcRemoverCutOff83Hz = 1U,
 kPDM_DcRemoverCutOff152Hz = 2U,
 kPDM_DcRemoverBypass = 3U }

PDM DC remover configurations.

- enum `_pdm_df_quality_mode` {

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` {

```

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` {

 `kPDM_HwvadHpfBypassed` = 0x0U,
`kPDM_HwvadHpfCutOffFreq1750Hz` = 0x1U,
`kPDM_HwvadHpfCutOffFreq215Hz` = 0x2U,
`kPDM_HwvadHpfCutOffFreq102Hz` = 0x3U }
- High pass filter configure cut-off frequency.*
- enum `_pdm_hwvad_filter_status` {

 `kPDM_HwvadInternalFilterNormalOperation` = 0U,
`kPDM_HwvadInternalFilterInitial` = PDM_VAD0_CTRL_1_VADST10_MASK }
- HWVAD internal filter status.*
- enum `_pdm_hwvad_zcd_result` {

 `kPDM_HwvadResultOREnergyBasedDetection`,
`kPDM_HwvadResultANDEnergyBasedDetection` }
- PDM voice activity detector zero cross detector result.*

Driver version

- #define `FSL_PDM_DRIVER_VERSION` (`MAKE_VERSION(2, 9, 1)`)
- Version 2.9.1.*

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 **PDMGetInstance** (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

- static int16_t **PDM_ReadData** (PDM_Type *base, uint32_t channel)
Reads data from the PDM FIFO.
- void **PDM_ReadNonBlocking** (PDM_Type *base, uint32_t startChannel, uint32_t channelNums, int16_t *buffer, size_t size)
PDM read data non blocking.
- 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_SetHvvadConfig** (PDM_Type *base, const pdm_hwvad_config_t *config)
Configure voice activity detector.
- static void **PDM_ForceHvvadOutputDisable** (PDM_Type *base, bool enable)
PDM hwvad force output disable.
- static void **PDM_ResetHvvad** (PDM_Type *base)
PDM hwvad reset.
- static void **PDM_EnableHvvad** (PDM_Type *base, bool enable)
Enable/Disable Voice activity detector.
- static void **PDM_EnableHvvadInterrupts** (PDM_Type *base, uint32_t mask)
Enables the PDM Voice Detector interrupt requests.
- static void **PDM_DisableHvvadInterrupts** (PDM_Type *base, uint32_t mask)
Disables the PDM Voice Detector interrupt requests.
- static void **PDM_ClearHvvadInterruptStatusFlags** (PDM_Type *base, uint32_t mask)
Clears the PDM voice activity detector status flags.
- static uint32_t **PDM_GetHvvadInterruptStatusFlags** (PDM_Type *base)
Clears the PDM voice activity detector status flags.
- static uint32_t **PDM_GetHvvadInitialFlag** (PDM_Type *base)
Get the PDM voice activity detector initial flags.

- static uint32_t **PDM_GetHvvadVoiceDetectedFlag** (PDM_Type *base)
Get the PDM voice activity detector voice detected flags.
- static void **PDM_EnableHvvadSignalFilter** (PDM_Type *base, bool enable)
Enables/disables voice activity detector signal filter.
- void **PDM_SetHvvadSignalFilterConfig** (PDM_Type *base, bool enableMaxBlock, uint32_t signalGain)
Configure voice activity detector signal filter.
- void **PDM_SetHvvadNoiseFilterConfig** (PDM_Type *base, const pdm_hvvad_noise_filter_t *config)
Configure voice activity detector noise filter.
- static void **PDM_EnableHvvadZeroCrossDetector** (PDM_Type *base, bool enable)
Enables/disables voice activity detector zero cross detector.
- void **PDM_SetHvvadZeroCrossDetectorConfig** (PDM_Type *base, const pdm_hvvad_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_SetHvvadInternalFilterStatus** (PDM_Type *base, pdm_hvvad_filter_status_t status)
set hvvad internal filter status .
- void **PDM_SetHvvadInEnvelopeBasedMode** (PDM_Type *base, const pdm_hvvad_config_t *hvvadConfig, const pdm_hvvad_noise_filter_t *noiseConfig, const pdm_hvvad_zero_cross_detector_t *zcdConfig, uint32_t signalGain)
set HWVAD in envelope based mode .
- void **PDM_SetHvvadInEnergyBasedMode** (PDM_Type *base, const pdm_hvvad_config_t *hvvadConfig, const pdm_hvvad_noise_filter_t *noiseConfig, const pdm_hvvad_zero_cross_detector_t *zcdConfig, uint32_t signalGain)
brief set HWVAD in energy based mode .
- void **PDM_EnableHvvadInterruptCallback** (PDM_Type *base, pdm_hvvad_callback_t vadCallback, void *userData, bool enable)
Enable/Disable hvvad 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

Data Fields

- `pdm_dc_remover_t cutOffFreq`
DC remover cut off frequency.
- `pdm_df_output_gain_t gain`
Decimation Filter Output Gain.

16.3.3.2 struct _pdm_config

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

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`
Voice activity detector input gain.
- `uint32_t frameTime`
Voice activity frame time.
- `pdm_hwvad_hpf_config_t cutOffFreq`
High pass filter cut off frequency.
- `bool enableFrameEnergy`
If frame energy enabled, true means enable.
- `bool enablePreFilter`
If pre-filter enabled.

Field Documentation

(1) `uint8_t _pdm_hwvad_config::initializeTime`

16.3.3.4 struct _pdm_hwvad_noise_filter

Data Fields

- `bool enableAutoNoiseFilter`
If noise filterer 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

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::enableAutoThreshold`

16.3.3.6 struct _pdm_transfer

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::data
- (2) volatile size_t _pdm_transfer::dataSize

16.3.3.7 struct _pdm_hwvad_notification

16.3.3.8 struct _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_CLK_LOW* PDM clock frequency low.
- 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_StatusFrequencyLow Mic app clock frequency not high enough.

kPDM_StatusCh0FifoDataAvailable channel 0 fifo data reached watermark level

kPDM_StatusCh1FifoDataAvailable channel 1 fifo data reached watermark level

kPDM_StatusCh2FifoDataAvailable channel 2 fifo data reached watermark level

kPDM_StatusCh3FifoDataAvailable channel 3 fifo data reached watermark level

kPDM_StatusCh4FifoDataAvailable channel 4 fifo data reached watermark level

kPDM_StatusCh5FifoDataAvailable channel 5 fifo data reached watermark level

kPDM_StatusCh6FifoDataAvailable channel 6 fifo data reached watermark level

kPDM_StatusCh7FifoDataAvailable channel 7 fifo data reached watermark level

16.3.4.4 enum _pdm_channel_enable_mask

Enumerator

kPDM_EnableChannel0 channge1 0 enable mask

kPDM_EnableChannel1 channge1 1 enable mask

kPDM_EnableChannel2 channge1 2 enable mask

kPDM_EnableChannel3 channge1 3 enable mask

kPDM_EnableChannel4 channge1 4 enable mask

kPDM_EnableChannel5 channge1 5 enable mask

kPDM_EnableChannel6 channge1 6 enable mask

kPDM_EnableChannel7 channge1 7 enable mask

16.3.4.5 enum _pdm_fifo_status

Enumerator

<i>kPDM_FifoStatusUnderflowCh0</i>	channel0 fifo status underflow
<i>kPDM_FifoStatusUnderflowCh1</i>	channel1 fifo status underflow
<i>kPDM_FifoStatusUnderflowCh2</i>	channel2 fifo status underflow
<i>kPDM_FifoStatusUnderflowCh3</i>	channel3 fifo status underflow
<i>kPDM_FifoStatusUnderflowCh4</i>	channel4 fifo status underflow
<i>kPDM_FifoStatusUnderflowCh5</i>	channel5 fifo status underflow
<i>kPDM_FifoStatusUnderflowCh6</i>	channel6 fifo status underflow
<i>kPDM_FifoStatusUnderflowCh7</i>	channel7 fifo status underflow
<i>kPDM_FifoStatusOverflowCh0</i>	channel0 fifo status overflow
<i>kPDM_FifoStatusOverflowCh1</i>	channel1 fifo status overflow
<i>kPDM_FifoStatusOverflowCh2</i>	channel2 fifo status overflow
<i>kPDM_FifoStatusOverflowCh3</i>	channel3 fifo status overflow
<i>kPDM_FifoStatusOverflowCh4</i>	channel4 fifo status overflow
<i>kPDM_FifoStatusOverflowCh5</i>	channel5 fifo status overflow
<i>kPDM_FifoStatusOverflowCh6</i>	channel6 fifo status overflow
<i>kPDM_FifoStatusOverflowCh7</i>	channel7 fifo status overflow

16.3.4.6 enum _pdm_output_status

Enumerator

<i>kPDM_OutputStatusUnderFlowCh0</i>	channel0 output status underflow
<i>kPDM_OutputStatusUnderFlowCh1</i>	channel1 output status underflow
<i>kPDM_OutputStatusUnderFlowCh2</i>	channel2 output status underflow
<i>kPDM_OutputStatusUnderFlowCh3</i>	channel3 output status underflow
<i>kPDM_OutputStatusUnderFlowCh4</i>	channel4 output status underflow
<i>kPDM_OutputStatusUnderFlowCh5</i>	channel5 output status underflow
<i>kPDM_OutputStatusUnderFlowCh6</i>	channel6 output status underflow
<i>kPDM_OutputStatusUnderFlowCh7</i>	channel7 output status underflow
<i>kPDM_OutputStatusOverFlowCh0</i>	channel0 output status overflow
<i>kPDM_OutputStatusOverFlowCh1</i>	channel1 output status overflow
<i>kPDM_OutputStatusOverFlowCh2</i>	channel2 output status overflow
<i>kPDM_OutputStatusOverFlowCh3</i>	channel3 output status overflow
<i>kPDM_OutputStatusOverFlowCh4</i>	channel4 output status overflow
<i>kPDM_OutputStatusOverFlowCh5</i>	channel5 output status overflow
<i>kPDM_OutputStatusOverFlowCh6</i>	channel6 output status overflow
<i>kPDM_OutputStatusOverFlowCh7</i>	channel7 output status overflow

16.3.4.7 enum _pdm_dc_remover

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

Enumerator

kPDM_QualityModeMedium quality mode memdium
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_qulaity_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

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

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

Enumerator

- kPDM_HwvadInternalFilterNormalOperation*** internal filter ready for normal operation
- kPDM_HwvadInternalFilterInitial*** interla filter are initial

16.3.4.16 enum _pdm_hwvad_zcd_result

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 superceded 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_StatusFrequencyLow and kPDM_StatusCh7FifoDataAvailable.

**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"> • kPDM_ErrorInterruptEnable • kPDM_FIFOInterruptEnable

**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"> • kPDM_ErrorInterruptEnable • kPDM_FIFOInterruptEnable

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>	Which data channel used.

Returns

data register address.

16.3.5.24 static int16_t PDM_ReadData (PDM_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

<i>base</i>	PDM base pointer.
<i>channel</i>	Data channel used.

Returns

Data in PDM FIFO.

16.3.5.25 void PDM_ReadNonBlocking (PDM_Type * *base*, uint32_t *startChannel*, uint32_t *channelNums*, int16_t * *buffer*, size_t *size*)

So the actually read data byte size in this function is (*size* * 2 * *channelNums*).

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 16bit data to read.

16.3.5.26 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.

**16.3.5.27 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.

16.3.5.28 void PDM_SetHvvadConfig (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 .

16.3.5.29 static void PDM_ForceHvvadOutputDisable (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.

16.3.5.30 static void PDM_ResetHvvad (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
-------------	------------------

16.3.5.31 static void PDM_EnableHvvad (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.

16.3.5.32 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"> • kPDM_HWVADErrorInterruptEnable • kPDM_HWVADIInterruptEnable

16.3.5.33 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"> • kPDM_HWVADErrorInterruptEnable • kPDM_HWVADIInterruptEnable

16.3.5.34 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.

16.3.5.35 static uint32_t PDM_GetHwvadInterruptStatusFlags (PDM_Type * *base*) [inline], [static]

Parameters

<i>base</i>	PDM base pointer
-------------	------------------

Returns

status, reference _pdm_hwvad_int_status

16.3.5.36 static uint32_t PDM_GetHwvadInitialFlag (PDM_Type * *base*) [inline], [static]

Parameters

<i>base</i>	PDM base pointer
-------------	------------------

Returns

initial flag.

16.3.5.37 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.

16.3.5.38 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.

16.3.5.39 void PDM_SetHwvadSignalFilterConfig (PDM_Type * *base*, bool *enableMaxBlock*, uint32_t *signalGain*)

Parameters

<i>base</i>	PDM base pointer
<i>enableMaxBlock</i>	If signal maximum block enabled.
<i>signalGain</i>	Gain value for the signal energy.

16.3.5.40 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 .

16.3.5.41 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.

16.3.5.42 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 .

16.3.5.43 static uint16_t PDM_GetNoiseData (PDM_Type * *base*) [inline], [static]

Parameters

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

Returns

Data in PDM noise data register.

16.3.5.44 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.

16.3.5.45 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.

16.3.5.46 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.

16.3.5.47 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

None.

16.3.5.48 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.

16.3.5.49 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.
--	---------------------

16.3.5.50 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 pdm_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_PDM_Busy</i>	Previous receive still not finished.

16.3.5.51 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.

16.3.5.52 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~(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](#)
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**

- **volatile uint8_t queueDriver**
Index for user to queue transfer.
- **Index for driver to get the transfer data and size.**

Field Documentation

- (1) **uint8_t _pdm_sdma_handle::nbytes**
- (2) **sdma_buffer_descriptor_t _pdm_sdma_handle::bdPool[PDM_XFER_QUEUE_SIZE]**
- (3) **pdm_transfer_t _pdm_sdma_handle::pdmQueue[PDM_XFER_QUEUE_SIZE]**
- (4) **volatile uint8_t _pdm_sdma_handle::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`
RDC hardware configuration. [More...](#)
- struct `_rdc_domain_assignment`
Master domain assignment. [More...](#)
- struct `_rdc_periph_access_config`
Peripheral domain access permission configuration. [More...](#)
- struct `_rdc_mem_access_config`
Memory region domain access control configuration. [More...](#)
- struct `_rdc_mem_status`
Memory region access violation status. [More...](#)

Typedefs

- typedef struct `_rdc_hardware_config` `rdc_hardware_config_t`
RDC hardware configuration.
- typedef struct `_rdc_domain_assignment` `rdc_domain_assignment_t`
Master domain assignment.
- typedef struct `_rdc_periph_access_config` `rdc_periph_access_config_t`
Peripheral domain access permission configuration.
- typedef struct `_rdc_mem_access_config` `rdc_mem_access_config_t`
Memory region domain access control configuration.
- typedef struct `_rdc_mem_status` `rdc_mem_status_t`
Memory region access violation status.

Enumerations

- enum `_rdc_interrupts` { `kRDC_RestoreCompleteInterrupt` = `RDC_INTCTRL_RCI_EN_MASK` }
RDC interrupts.
- enum `_rdc_flags` { `kRDC_PowerDownDomainOn` = `RDC_STAT_PDS_MASK` }

- enum `_rdc_access_policy` {

 `kRDC_NoAccess` = 0,
 `kRDC_WriteOnly` = 1,
 `kRDC_ReadOnly` = 2,
 `kRDC_ReadWrite` = 3 }

 Access permission policy.

Functions

- void `RDC_Init` (`RDC_Type` *base)

 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)

- void [RDC_GetMemViolationStatus](#) (RDC_Type *base, rdc_mem_t mem, rdc_mem_status_t *status)
Enable or disable memory access policy configuration.
- static void [RDC_ClearMemViolationFlag](#) (RDC_Type *base, rdc_mem_t mem)
Get the memory region violation status.
- static uint8_t [RDC_GetMemAccessPolicy](#) (RDC_Type *base, rdc_mem_t mem, uint8_t domainId)
Clear the memory region violation flag.
- static uint8_t [RDC_GetCurrentMasterDomainId](#) (RDC_Type *base)
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

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::domainNumber**
- (2) **uint32_t _rdc_hardware_config::masterNumber**
- (3) **uint32_t _rdc_hardware_config::periphNumber**
- (4) **uint32_t _rdc_hardware_config::memNumber**

17.2.2 struct _rdc_domain_assignment

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::domainId`
- (2) `uint32_t _rdc_domain_assignment::__pad0__`
- (3) `uint32_t _rdc_domain_assignment::lock`

17.2.3 struct _rdc_periph_access_config**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::periph`
- (2) `bool _rdc_periph_access_config::lock`
- (3) `bool _rdc_periph_access_config::enableSema`
- (4) `uint16_t _rdc_periph_access_config::policy`

17.2.4 struct _rdc_mem_access_config

Note that when setting the `rdc_mem_access_config_t::baseAddress` and `rdc_mem_access_config_t::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::mem`
- (2) `bool _rdc_mem_access_config::lock`
- (3) `uint64_t _rdc_mem_access_config::baseAddress`
- (4) `uint64_t _rdc_mem_access_config::endAddress`
- (5) `uint16_t _rdc_mem_access_config::policy`

17.2.5 struct _rdc_mem_status

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::hasViolation`
- (2) `uint8_t _rdc_mem_status::domainID`
- (3) `uint64_t _rdc_mem_status::address`

17.3 Typedef Documentation

17.3.1 typedef struct _rdc_mem_access_config rdc_mem_access_config_t

Note that when setting the `rdc_mem_access_config_t::baseAddress` and `rdc_mem_access_config_t::endAddress`, should be aligned to the region resolution, see `rdc_mem_t` definitions.

17.4 Enumeration Type Documentation

17.4.1 enum _rdc_interrupts

Enumerator

kRDC_RestoreCompleteInterrupt Interrupt generated when the RDC has completed restoring state to a recently re-powered memory regions.

17.4.2 enum _rdc_flags

Enumerator

kRDC_PowerDownDomainOn Power down domain is ON.

17.4.3 enum _rdc_access_policy

Enumerator

kRDC_NoAccess Could not read or write.

kRDC_WriteOnly Write only.

kRDC_ReadOnly Read only.

kRDC_ReadWrite Read and write.

17.5 Function Documentation

17.5.1 void RDC_Init (RDC_Type * *base*)

This function enables the RDC clock.

Parameters

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

17.5.2 void RDC_Deinit (RDC_Type * *base*)

This function disables the RDC clock.

Parameters

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

17.5.3 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.

17.5.4 static void RDC_EnableInterrupts (**RDC_Type** * *base*, **uint32_t** *mask*) [**inline**], [**static**]

Parameters

<i>base</i>	RDC peripheral base address.
<i>mask</i>	Interrupts to enable, it is OR'ed value of enum _rdc_interrupts .

17.5.5 static void RDC_DisableInterrupts (**RDC_Type** * *base*, **uint32_t** *mask*) [**inline**], [**static**]

Parameters

<i>base</i>	RDC peripheral base address.
<i>mask</i>	Interrupts to disable, it is OR'ed value of enum _rdc_interrupts .

17.5.6 static **uint32_t** RDC_GetInterruptStatus (**RDC_Type** * *base*) [**inline**], [**static**]

Parameters

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

Returns

Interrupts pending status, it is OR'ed value of enum [_rdc_interrupts](#).

17.5.7 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 _rdc_interrupts .

17.5.8 static uint32_t RDC_GetStatus (RDC_Type * *base*) [inline], [static]

Parameters

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

Returns

mask RDC status, it is OR'ed value of enum [_rdc_flags](#).

17.5.9 static void RDC_ClearStatus (RDC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	RDC peripheral base address.
<i>mask</i>	RDC status to clear, it is OR'ed value of enum _rdc_flags .

17.5.10 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>domainAssignment</i>	Pointer to the assignment.

17.5.11 void RDC_GetDefaultMasterDomainAssignment (rdc_domain_assignment_t * *domainAssignment*)

The default configuration is:

```
assignment->domainId = 0U;
assignment->lock = 0U;
```

Parameters

<i>domainAssignment</i>	Pointer to the assignment.
-------------------------	----------------------------

17.5.12 static void RDC_LockMasterDomainAssignment (RDC_Type * *base*, rdc_master_t *master*) [inline], [static]

Once locked, it could not be unlocked until next reset.

Parameters

<i>base</i>	RDC peripheral base address.
<i>master</i>	Which master to lock.

17.5.13 void RDC_SetPeriphAccessConfig (RDC_Type * *base*, const rdc_periph_access_config_t * *config*)

Parameters

<i>base</i>	RDC peripheral base address.
<i>config</i>	Pointer to the policy configuration.

17.5.14 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.
---------------	--------------------------------------

17.5.15 static void RDC_LockPeriphAccessConfig (**RDC_Type** * *base*, **rdc_periph_t** *periph*) [inline], [static]

Once locked, it could not be unlocked until reset.

Parameters

<i>base</i>	RDC peripheral base address.
<i>periph</i>	Which peripheral to lock.

17.5.16 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](#).

17.5.17 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.

17.5.18 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.
---------------	--------------------------------------

17.5.19 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.

17.5.20 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.

17.5.21 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.

17.5.22 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.

17.5.23 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](#).

17.5.24 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 RD-C_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`
`SAI user configuration structure.` [More...](#)
- struct `_sai_transfer_format`
`sai transfer format` [More...](#)
- struct `_sai_master_clock`
`master clock configurations` [More...](#)
- struct `_sai_fifo`
`sai fifo configurations` [More...](#)
- struct `_sai_bit_clock`
`sai bit clock configurations` [More...](#)
- struct `_sai_frame_sync`
`sai frame sync configurations` [More...](#)
- struct `_sai_serial_data`
`sai serial data configurations` [More...](#)
- struct `_sai_transceiver`
`sai transceiver configurations` [More...](#)
- struct `_sai_transfer`
`SAI transfer structure.` [More...](#)
- struct `_sai_handle`
`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` 1
`sai fifo feature`

Typedefs

- typedef enum `_sai_protocol` `sai_protocol_t`
`Define the SAI bus type.`
- typedef enum `_sai_master_slave` `sai_master_slave_t`
`Master or slave mode.`
- typedef enum `_sai_mono_stereo` `sai_mono_stereo_t`
`Mono or stereo audio format.`
- typedef enum `_sai_data_order` `sai_data_order_t`
`SAI data order, MSB or LSB.`
- typedef enum `_sai_clock_polarity` `sai_clock_polarity_t`
`SAI clock polarity, active high or low.`
- typedef enum `_sai_sync_mode` `sai_sync_mode_t`
`Synchronous or asynchronous mode.`

- `typedef enum _sai_bclk_source sai_bclk_source_t`
Bit clock source.
- `typedef enum _sai_reset_type sai_reset_type_t`
The reset type.
- `typedef enum _sai_fifo_packing sai_fifo_packing_t`
The SAI packing mode The mode includes 8 bit and 16 bit packing.
- `typedef struct _sai_config sai_config_t`
SAI user configuration structure.
- `typedef enum _sai_sample_rate sai_sample_rate_t`
Audio sample rate.
- `typedef enum _sai_word_width sai_word_width_t`
Audio word width.
- `typedef enum _sai_data_pin_state sai_data_pin_state_t`
sai data pin state definition
- `typedef enum _sai_fifo_combine sai_fifo_combine_t`
sai fifo combine mode definition
- `typedef enum _sai_transceiver_type sai_transceiver_type_t`
sai transceiver type
- `typedef enum _sai_frame_sync_len sai_frame_sync_len_t`
sai frame sync len
- `typedef struct _sai_transfer_format sai_transfer_format_t`
sai transfer format
- `typedef struct _sai_master_clock sai_master_clock_t`
master clock configurations
- `typedef struct _sai_fifo sai_fifo_t`
sai fifo configurations
- `typedef struct _sai_bit_clock sai_bit_clock_t`
sai bit clock configurations
- `typedef struct _sai_frame_sync sai_frame_sync_t`
sai frame sync configurations
- `typedef struct _sai_serial_data sai_serial_data_t`
sai serial data configurations
- `typedef struct _sai_transceiver sai_transceiver_t`
sai transceiver configurations
- `typedef struct _sai_transfer sai_transfer_t`
SAI transfer structure.
- `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) }

• 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 {
 kSAI_BusLeftJustified = 0x0U,
 kSAI_BusRightJustified,
 kSAI_BusI2S,
 kSAI_BusPCMA,
 kSAI_BusPCMB }

 Define the SAI bus type.

• enum _sai_master_slave {
 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 {
 kSAI_Stereo = 0x0U,
 kSAI_MonoRight,
 kSAI_MonoLeft }

 Mono or stereo audio format.

• enum _sai_data_order {
 kSAI_DataLSB = 0x0U,
 kSAI_DataMSB }

 SAI data order, MSB or LSB.

• enum _sai_clock_polarity {
 kSAI_PolarityActiveHigh = 0x0U,
 kSAI_PolarityActiveLow = 0x1U,
 kSAI_SampleOnFallingEdge = 0x0U,
 kSAI_SampleOnRisingEdge = 0x1U }

 SAI clock polarity, active high or low.

• enum _sai_sync_mode {
 kSAI_ModeAsync = 0x0U,
 kSAI_ModeSync }

 Synchronous or asynchronous mode.

• enum _sai_bclk_source {

```

```
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,
 kSAI\_FIFORequestInterruptEnable = I2S\_TCSR\_FRIE\_MASK }
   
*\_sai\_interrupt\_enable\_t, The SAI interrupt enable flag*
- enum {
   
kSAI\_FIFOWarningDMAEnable = I2S\_TCSR\_FWDE\_MASK,
 kSAI\_FIFORequestDMAEnable = I2S\_TCSR\_FRDE\_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\_FIFORequestFlag = I2S\_TCSR\_FRF\_MASK,
 kSAI\_FIFOWarningFlag = I2S\_TCSR\_FWF\_MASK }
   
*\_sai\_flags, The SAI status flag*
- enum *\_sai\_reset\_type* {
   
kSAI\_ResetTypeSoftware = I2S\_TCSR\_SR\_MASK,
 kSAI\_ResetTypeFIFO = I2S\_TCSR\_FR\_MASK,
 kSAI\_ResetAll = I2S\_TCSR\_SR\_MASK | I2S\_TCSR\_FR\_MASK }
   
*The reset type.*
- enum *\_sai\_fifo\_packing* {
   
kSAI\_FifoPackingDisabled = 0x0U,
 kSAI\_FifoPacking8bit = 0x2U,
 kSAI\_FifoPacking16bit = 0x3U }
   
*The SAI packing mode The mode includes 8 bit and 16 bit packing.*
- enum *\_sai\_sample\_rate* {

```

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` {
   
kSAI\_WordWidth8bits = 8U,
   
kSAI\_WordWidth16bits = 16U,
   
kSAI\_WordWidth24bits = 24U,
   
kSAI\_WordWidth32bits = 32U }

*Audio word width.*

- enum `_sai_data_pin_state` {
   
kSAI\_DataPinStateTriState,
   
kSAI\_DataPinStateOutputZero = 1U }
- enum `_sai_fifo_combine` {
   
kSAI\_FifoCombineDisabled = 0U,
   
kSAI\_FifoCombineModeEnabledOnRead,
   
kSAI\_FifoCombineModeEnabledOnWrite,
   
kSAI\_FifoCombineModeEnabledOnReadWrite }
- enum `_sai_transceiver_type` {
   
kSAI\_Transmitter = 0U,
   
kSAI\_Receiver = 1U }
- enum `_sai_frame_sync_len` {
   
kSAI\_FrameSyncLenOneBitClk = 0U,
   
kSAI\_FrameSyncLenPerWordWidth = 1U }

*sai frame sync len*

## Driver version

- #define `FSL_SAI_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 2)`)  
*Version 2.4.2.*

## Initialization and deinitialization

- 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\_SetMasterClockConfig** (I2S\_Type \*base, **sai\_master\_clock\_t** \*config)  
*Master clock configurations.*
- void **SAI\_TxSetFifoConfig** (I2S\_Type \*base, **sai\_fifo\_t** \*config)  
*SAI transmitter fifo configurations.*
- void **SAI\_RxSetFifoConfig** (I2S\_Type \*base, **sai\_fifo\_t** \*config)  
*SAI receiver fifo 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 Rx status flag state.*
- static void **SAI\_RxClearStatusFlags** (I2S\_Type \*base, uint32\_t mask)  
*Clears the SAI Rx status flag state.*
- void **SAI\_TxSoftwareReset** (I2S\_Type \*base, **sai\_reset\_type\_t** 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.
- void **SAI\_TxSetFIFOPacking** (I2S\_Type \*base, sai\_fifo\_packing\_t pack)  
    *Set Tx FIFO packing feature.*
- void **SAI\_RxSetFIFOPacking** (I2S\_Type \*base, sai\_fifo\_packing\_t pack)  
    *Set Rx FIFO packing feature.*
- static void **SAI\_TxSetFIFOErrorContinue** (I2S\_Type \*base, bool isEnabled)  
    *Set Tx FIFO error continue.*
- static void **SAI\_RxSetFIFOErrorContinue** (I2S\_Type \*base, bool isEnabled)  
    *Set Rx FIFO error continue.*

## Interrupts

- static void **SAI\_TxEnableInterrupts** (I2S\_Type \*base, uint32\_t mask)  
    *Enables the SAI Tx interrupt requests.*
- static void **SAI\_RxEnableInterrupts** (I2S\_Type \*base, uint32\_t mask)  
    *Enables the SAI Rx interrupt requests.*
- 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\_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\_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)  
*Rx interrupt handler.*

## 19.5.2 Data Structure Documentation

### 19.5.2.1 struct \_sai\_config

#### Data Fields

- sai\_protocol\_t protocol  
*Audio bus protocol in SAI.*

- **sai\_sync\_mode\_t syncMode**  
*SAI sync mode, control Tx/Rx clock sync.*
- **bool mclkOutputEnable**  
*Master clock output enable, true means master clock divider enabled.*
- **sai\_bclk\_source\_t bclkSource**  
*Bit Clock source.*
- **sai\_master\_slave\_t masterSlave**  
*Master or slave.*

### 19.5.2.2 struct \_sai\_transfer\_format

#### 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 watermark**  
*Watermark value.*
- **uint8\_t channel**  
*Transfer start channel.*
- **uint8\_t channelMask**  
*enabled channel mask value, reference \_sai\_channel\_mask*
- **uint8\_t endChannel**  
*end channel number*
- **uint8\_t channelNums**  
*Total enabled channel numbers.*
- **sai\_protocol\_t protocol**  
*Which audio protocol used.*
- **bool isFrameSyncCompact**  
*True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.*

#### Field Documentation

(1) **bool \_sai\_transfer\_format::isFrameSyncCompact**

### 19.5.2.3 struct \_sai\_master\_clock

#### Data Fields

- **bool mclkOutputEnable**  
*master clock output enable*
- **uint32\_t mclkHz**  
*target mclk frequency*
- **uint32\_t mclkSourceClkHz**  
*mclk source frequency*

#### 19.5.2.4 struct \_sai\_fifo

##### Data Fields

- bool `fifoContinueOneError`  
*fifo continues when error occur*
- `sai_fifo_combine_t fifoCombine`  
*fifo combine mode*
- `sai_fifo_packing_t fifoPacking`  
*fifo packing mode*
- uint8\_t `fifoWatermark`  
*fifo watermark*

#### 19.5.2.5 struct \_sai\_bit\_clock

##### 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::bclkInputDelay`

#### 19.5.2.6 struct \_sai\_frame\_sync

##### 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.*
- bool `frameSyncGenerateOnDemand`  
*internal frame sync is generated when FIFO waring flag is clear*
- `sai_clock_polarity_t frameSyncPolarity`  
*frame sync polarity*

### 19.5.2.7 struct \_sai\_serial\_data

#### Data Fields

- **sai\_data\_pin\_state\_t dataMode**  
*sai data pin state when slots masked or channel disabled*
- **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*

### 19.5.2.8 struct \_sai\_transceiver

#### 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\_fifo\_t fifo**  
*fifo 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.*

### 19.5.2.9 struct \_sai\_transfer

#### Data Fields

- `uint8_t * data`  
*Data start address to transfer.*
- `size_t dataSize`  
*Transfer size.*

#### Field Documentation

- (1) `uint8_t* _sai_transfer::data`
- (2) `size_t _sai_transfer::dataSize`

### 19.5.2.10 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.*
- `uint8_t watermark`  
*Watermark value.*

### 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

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

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

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

Enumerator

***kSAI\_DataLSB*** LSB bit transferred first.

***kSAI\_DataMSB*** MSB bit transferred first.

#### 19.5.4.7 enum \_sai\_clock\_polarity

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

Enumerator

***kSAI\_ModeAsync*** Asynchronous mode.

***kSAI\_ModeSync*** Synchronous mode (with receiver or transmit)

#### 19.5.4.9 enum \_sai\_bclk\_source

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.

#### 19.5.4.10 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.
- kSAI\_FIFORequestInterruptEnable* FIFO request, means reached watermark.

#### 19.5.4.11 anonymous enum

Enumerator

- kSAI\_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.
- kSAI\_FIFORequestDMAEnable* FIFO request caused by the DMA request.

#### 19.5.4.12 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\_FIFORequestFlag* FIFO request flag.
- kSAI\_FIFOWarningFlag* FIFO warning flag.

#### 19.5.4.13 enum \_sai\_reset\_type

Enumerator

- kSAI\_ResetTypeSoftware* Software reset, reset the logic state.
- kSAI\_ResetTypeFIFO* FIFO reset, reset the FIFO read and write pointer.

***kSAI\_ResetAll*** All reset.

#### 19.5.4.14 enum \_sai\_fifo\_packing

Enumerator

***kSAI\_FifoPackingDisabled*** Packing disabled.

***kSAI\_FifoPacking8bit*** 8 bit packing enabled

***kSAI\_FifoPacking16bit*** 16bit packing enabled

#### 19.5.4.15 enum \_sai\_sample\_rate

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

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\_data\_pin\_state

Enumerator

***kSAI\_DataPinStateTriState*** transmit data pins are tri-stated when slots are masked or channels are disabled

***kSAI\_DataPinStateOutputZero*** transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

#### 19.5.4.18 enum \_sai\_fifo\_combine

Enumerator

*kSAI\_FifoCombineDisabled* sai fifo combine mode disabled  
*kSAI\_FifoCombineModeEnabledOnRead* sai fifo combine mode enabled on FIFO reads  
*kSAI\_FifoCombineModeEnabledOnWrite* sai fifo combine mode enabled on FIFO write  
*kSAI\_FifoCombineModeEnabledOnReadWrite* sai fifo combined mode enabled on FIFO read/writes

#### 19.5.4.19 enum \_sai\_transceiver\_type

Enumerator

*kSAI\_Transmitter* sai transmitter  
*kSAI\_Receiver* sai receiver

#### 19.5.4.20 enum \_sai\_frame\_sync\_len

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\_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. |
|-------------|-------------------|

#### 19.5.5.2 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. |
|-------------|-------------------|

#### 19.5.5.3 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 |
|-------------|------------------|

#### 19.5.5.4 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 |
|-------------|------------------|

#### 19.5.5.5 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. |

#### 19.5.5.6 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. |

#### 19.5.5.7 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. |

#### 19.5.5.8 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. |

#### 19.5.5.9 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. |

#### 19.5.5.10 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. |

#### 19.5.5.11 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.      |

**19.5.5.12 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.      |

**19.5.5.13 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. |

**19.5.5.14 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. |

#### 19.5.5.15 void SAI\_SetMasterClockConfig ( I2S\_Type \* *base*, sai\_master\_clock\_t \* *config* )

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer.            |
| <i>config</i> | master clock configurations. |

#### 19.5.5.16 void SAI\_TxSetFifoConfig ( I2S\_Type \* *base*, sai\_fifo\_t \* *config* )

Parameters

|               |                      |
|---------------|----------------------|
| <i>base</i>   | SAI base pointer.    |
| <i>config</i> | fifo configurations. |

#### 19.5.5.17 void SAI\_RxSetFifoConfig ( I2S\_Type \* *base*, sai\_fifo\_t \* *config* )

Parameters

|               |                      |
|---------------|----------------------|
| <i>base</i>   | SAI base pointer.    |
| <i>config</i> | fifo configurations. |

#### 19.5.5.18 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. |

**19.5.5.19 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. |

**19.5.5.20 void SAI\_TxSetSerialDataConfig ( I2S\_Type \* *base*, sai\_serial\_data\_t \* *config* )**

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | serial data configurations. |

**19.5.5.21 void SAI\_RxSetSerialDataConfig ( I2S\_Type \* *base*, sai\_serial\_data\_t \* *config* )**

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | serial data configurations. |

**19.5.5.22 void SAI\_TxSetConfig ( I2S\_Type \* *base*, sai\_transceiver\_t \* *config* )**

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>base</i>   | SAI base pointer.           |
| <i>config</i> | transmitter configurations. |

#### 19.5.5.23 void SAI\_RxSetConfig ( I2S\_Type \* *base*, sai\_transceiver\_t \* *config* )

Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>config</i> | receiver configurations. |

#### 19.5.5.24 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>saiChannel-<br/>Mask</i> | mask value of the channel to be enable. |

#### 19.5.5.25 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-<br/>Mask</i> | mask value of the channel to be enable. |

#### 19.5.5.26 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. |

**19.5.5.27 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. |

**19.5.5.28 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>frameSyncWidth</i> | length of frame sync.                |
| <i>bitWidth</i>       | audio data bitWidth.                 |
| <i>mode</i>           | audio data channel.                  |
| <i>saiChannelMask</i> | mask value of the channel to enable. |

**19.5.5.29 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.

**19.5.5.30 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> |

**19.5.5.31 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.

#### 19.5.5.32 static void SAI\_RxClearStatusFlags ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                   |
| <i>mask</i> | <p>State mask. It can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_WordStartFlag</li> <li>• kSAI_SyncErrorFlag</li> <li>• kSAI_FIFOErrorFlag</li> </ul> |

#### 19.5.5.33 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 |

#### 19.5.5.34 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 |

#### 19.5.5.35 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. |

#### 19.5.5.36 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. |

#### 19.5.5.37 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 |

#### 19.5.5.38 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 |

```
19.5.5.39 void SAI_TxSetBitClockPolarity (I2S_Type * base, sai_clock_polarity_t polarity)
```

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**19.5.5.40 void SAI\_RxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**19.5.5.41 void SAI\_TxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**19.5.5.42 void SAI\_RxSetFrameSyncPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

|                 |                  |
|-----------------|------------------|
| <i>base</i>     | SAI base pointer |
| <i>polarity</i> |                  |

**19.5.5.43 void SAI\_TxSetFIFOPacking ( I2S\_Type \* *base*, sai\_fifo\_packing\_t *pack* )**

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | SAI base pointer.                                    |
| <i>pack</i> | FIFO pack type. It is element of sai_fifo_packing_t. |

19.5.5.44 **void SAI\_RxSetFIFOPacking ( I2S\_Type \* *base*, sai\_fifo\_packing\_t *pack* )**

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | SAI base pointer.                                    |
| <i>pack</i> | FIFO pack type. It is element of sai_fifo_packing_t. |

#### 19.5.5.45 static void SAI\_TxSetFIFOErrorContinue ( I2S\_Type \* *base*, bool *isEnabled* ) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                       |
| <i>isEnabled</i> | Is FIFO error continue enabled, true means enable, false means disable. |

#### 19.5.5.46 static void SAI\_RxSetFIFOErrorContinue ( I2S\_Type \* *base*, bool *isEnabled* ) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| <i>base</i>      | SAI base pointer.                                                       |
| <i>isEnabled</i> | Is FIFO error continue enabled, true means enable, false means disable. |

#### 19.5.5.47 static void SAI\_TxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                                                                                                                                                                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                   |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**19.5.5.48 static void SAI\_RxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                          |
| <i>mask</i> | <p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**19.5.5.49 static void SAI\_TxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                          |
| <i>mask</i> | <p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

**19.5.5.50 static void SAI\_RxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                                                                                                                                                                          |
| <i>mask</i> | <p>interrupt source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_WordStartInterruptEnable</li> <li>• kSAI_SyncErrorInterruptEnable</li> <li>• kSAI_FIFOWarningInterruptEnable</li> <li>• kSAI_FIFORequestInterruptEnable</li> <li>• kSAI_FIFOErrorInterruptEnable</li> </ul> |

#### 19.5.5.51 static void SAI\_TxEnableDMA ( I2S\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                                                                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer                                                                                                                                                                                           |
| <i>mask</i>   | <p>DMA source The parameter can be combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_FIFOWarningDMAEnable</li> <li>• kSAI_FIFORequestDMAEnable</li> </ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.                                                                                                                                                            |

#### 19.5.5.52 static void SAI\_RxEnableDMA ( I2S\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]

Parameters

|             |                                                                                                                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SAI base pointer                                                                                                                                                                                             |
| <i>mask</i> | <p>DMA source The parameter can be a combination of the following sources if defined.</p> <ul style="list-style-type: none"> <li>• kSAI_FIFOWarningDMAEnable</li> <li>• kSAI_FIFORequestDMAEnable</li> </ul> |

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>enable</i> | True means enable DMA, false means disable DMA. |
|---------------|-------------------------------------------------|

### 19.5.5.53 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.

### 19.5.5.54 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.

### 19.5.5.55 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.                                     |

**19.5.5.56 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.                                     |

**19.5.5.57 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. |

**19.5.5.58 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.                                        |

**19.5.5.59 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.                                        |

**19.5.5.60 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.

#### 19.5.5.61 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 |

#### 19.5.5.62 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. |

#### 19.5.5.63 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. |

#### 19.5.5.64 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. |

#### 19.5.5.65 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 sai_transfer_t structure.                               |

Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Successfully started the data receive. |
| <i>kStatus_SAI_TxBusy</i>      | Previous receive still not finished.   |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.        |

### 19.5.5.66 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.        |

### 19.5.5.67 status\_t SAI\_TransferGetSendCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count sent.                                                      |

Return values

|                                      |                                                                |
|--------------------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>               | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferIn-Progress</i> | There is not a non-blocking transaction currently in progress. |

### 19.5.5.68 status\_t SAI\_TransferGetReceiveCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count received.                                                  |

Return values

|                                      |                                                                |
|--------------------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>               | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferIn-Progress</i> | There is not a non-blocking transaction currently in progress. |

#### 19.5.5.69 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. |

#### 19.5.5.70 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. |

#### 19.5.5.71 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. |

#### 19.5.5.72 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. |

#### 19.5.5.73 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. |

#### 19.5.5.74 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~(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`  
*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, 6, 0))`  
*Version 2.6.0.*

## 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.*
- 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::bdPool[SAI_XFER_QUEUE_SIZE]`
- (2) `sai_transfer_t _sai_sdma_handle::saiQueue[SAI_XFER_QUEUE_SIZE]`
- (3) `volatile uint8_t _sai_sdma_handle::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 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.           |

#### 19.6.4.4 status\_t SAI\_TransferReceiveSDMA ( I2S\_Type \* *base*, sai\_sdma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

Note

This interface returns immediately after the transfer initiates. Call the SAI\_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer                   |
| <i>handle</i> | SAI SDMA handle pointer.           |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI SDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.         |
| <i>kStatus_RxBusy</i>          | SAI is busy receiving data.            |

#### 19.6.4.5 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. |
|---------------|--------------------------|

#### 19.6.4.6 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. |

#### 19.6.4.7 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. |

#### 19.6.4.8 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. |

#### 19.6.4.9 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.

#### 19.6.4.10 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`  
*SDMA global configuration structure. [More...](#)*
- struct `_sdma_multi_fifo_config`  
*SDMA multi fifo configurations. [More...](#)*
- struct `_sdma_sw_done_config`  
*SDMA sw done configurations. [More...](#)*
- struct `_sdma_p2p_config`  
*SDMA peripheral to peripheral R7 config. [More...](#)*
- struct `_sdma_transfer_config`  
*SDMA transfer configuration. [More...](#)*
- struct `_sdma_buffer_descriptor`  
*SDMA buffer descriptor structure. [More...](#)*
- struct `_sdma_channel_control`  
*SDMA channel control descriptor structure. [More...](#)*
- struct `_sdma_context_data`  
*SDMA context structure for each channel. [More...](#)*
- struct `_sdma_handle`  
*SDMA transfer handle structure. [More...](#)*

### Typedefs

- typedef enum `_sdma_transfer_size` `sdma_transfer_size_t`  
*SDMA transfer configuration.*
- typedef enum `_sdma_bd_status` `sdma_bd_status_t`  
*SDMA buffer descriptor status.*
- typedef enum `_sdma_bd_command` `sdma_bd_command_t`  
*SDMA buffer descriptor command.*
- typedef enum  
  `_sdma_context_switch_mode` `sdma_context_switch_mode_t`

- *SDMA context switch mode.*
- `typedef enum _sdma_clock_ratio sdma_clock_ratio_t`  
*SDMA core clock frequency ratio to the ARM DMA interface.*
- `typedef enum _sdma_transfer_type sdma_transfer_type_t`  
*SDMA transfer type.*
- `typedef enum sdma_peripheral sdma_peripheral_t`  
*Peripheral type use SDMA.*
- `typedef enum _sdma_done_src sdma_done_src_t`  
*SDMA done source.*
- `typedef struct _sdma_config sdma_config_t`  
*SDMA global configuration structure.*
- `typedef struct _sdma_multi_fifo_config sdma_multi_fifo_config_t`  
*SDMA multi fifo configurations.*
- `typedef struct _sdma_sw_done_config sdma_sw_done_config_t`  
*SDMA sw done configurations.*
- `typedef struct _sdma_p2p_config sdma_p2p_config_t`  
*SDMA peripheral to peripheral R7 config.*
- `typedef struct _sdma_transfer_config sdma_transfer_config_t`  
*SDMA transfer configuration.*
- `typedef struct _sdma_buffer_descriptor sdma_buffer_descriptor_t`  
*SDMA buffer descriptor structure.*
- `typedef struct _sdma_channel_control sdma_channel_control_t`  
*SDMA channel control descriptor structure.*
- `typedef struct _sdma_context_data sdma_context_data_t`  
*SDMA context structure for each channel.*
- `typedef void(* sdma_callback )(struct _sdma_handle *handle, void *userData, bool transferDone, uint32_t bdIndex)`  
*Define callback function for SDMA.*
- `typedef struct _sdma_handle sdma_handle_t`  
*SDMA transfer handle structure.*

## Enumerations

- `enum _sdma_transfer_size {`  
 `kSDMA_TransferSize1Bytes = 0x1U,`  
 `kSDMA_TransferSize2Bytes = 0x2U,`  
 `kSDMA_TransferSize3Bytes = 0x3U,`  
 `kSDMA_TransferSize4Bytes = 0x0U }`  
*SDMA transfer configuration.*
- `enum _sdma_bd_status {`

```

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` {

```

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` {

```

kSDMA_ContextSwitchModeStatic = 0x0U,
kSDMA_ContextSwitchModeDynamicLowPower,
kSDMA_ContextSwitchModeDynamicWithNoLoop,
kSDMA_ContextSwitchModeDynamic }

```

*SDMA context switch mode.*

- enum `_sdma_clock_ratio` {

```

kSDMA_HalfARMClockFreq = 0x0U,
kSDMA_ARMClockFreq }

```

*SDMA core clock frequency ratio to the ARM DMA interface.*

- enum `_sdma_transfer_type` {

```

kSDMA_MemoryToMemory = 0x0U,
kSDMA_PeripheralToMemory,
kSDMA_MemoryToPeripheral,
kSDMA_PeripheralToPeripheral }

```

*SDMA transfer type.*

- enum `sdma_peripheral` {

```

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 = 0xFFFF,  
  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 {

```

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, 2))  
*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 channelMask)  
*Set SDMA request source mapping channel.*
- static void **SDMA\_StartChannelSoftware** (SDMAARM\_Type \*base, uint32\_t channel)  
*Start a SDMA channel by software trigger.*
- static void **SDMA\_StartChannelEvents** (SDMAARM\_Type \*base, uint32\_t channel)  
*Start a SDMA channel by hardware events.*
- static void **SDMA\_StopChannel** (SDMAARM\_Type \*base, uint32\_t channel)  
*Stop a SDMA channel.*
- 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\_PrepTransfer** (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\_PrepP2PTransfer** (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.*
- `bool SDMA_IsPeripheralInSPBA (uint32_t addr)`  
*Judge if address located in SPBA.*
- `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

#### 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::enableRealTimeDebugPin`
- (2) `bool _sdma_config::isSoftwareResetClearLock`

### 20.3.2 struct \_sdma\_multi\_fifo\_config

#### 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

#### 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

#### 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::continuousTransfer**

### 20.3.5 struct \_sdma\_transfer\_config

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::srcTransferSize**
- (2) **sdma\_transfer\_size\_t \_sdma\_transfer\_config::destTransferSize**
- (3) **uint32\_t \_sdma\_transfer\_config::scriptAddr**
- (4) **uint32\_t \_sdma\_transfer\_config::eventSource**  
0 means no event, use software trigger
- (5) **sdma\_transfer\_type\_t \_sdma\_transfer\_config::type**

### 20.3.6 struct \_sdma\_buffer\_descriptor

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::count`
- (2) `uint32_t _sdma_buffer_descriptor::bufferAddr`
- (3) `uint32_t _sdma_buffer_descriptor::extendBufferAddr`

**20.3.7 struct \_sdma\_channel\_control****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**

This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

**Data Fields**

- `uint32_t GeneralReg [8]`  
*8 general registers used for SDMA RISC core*

**20.3.9 struct \_sdma\_handle****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`

- `uint32_t eventSource`  
*How many buffer descriptor.*
- `uint32_t eventSource1`  
*Event source count for the channel.*
- `sdma_context_data_t * context`  
*Channel context to execute in SDMA.*
- `uint8_t channel`  
*SDMA channel number.*
- `uint8_t priority`  
*SDMA channel priority.*
- `uint8_t flags`  
*The status of the current channel.*

## Field Documentation

- (1) `sdma_callback _sdma_handle::callback`
- (2) `void* _sdma_handle::userData`
- (3) `SDMAARM_Type* _sdma_handle::base`
- (4) `sdma_buffer_descriptor_t* _sdma_handle::BDPool`
- (5) `uint8_t _sdma_handle::channel`
- (6) `uint8_t _sdma_handle::flags`

## 20.4 Macro Definition Documentation

### 20.4.1 `#define FSL_SDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 2))`

Version 2.4.2.

## 20.5 Typedef Documentation

### 20.5.1 `typedef enum _sdma_clock_ratio sdma_clock_ratio_t`

### 20.5.2 `typedef struct _sdma_config sdma_config_t`

### 20.5.3 `typedef struct _sdma_multi_fifo_config sdma_multi_fifo_config_t`

### 20.5.4 `typedef struct _sdma_sw_done_config sdma_sw_done_config_t`

### 20.5.5 `typedef struct _sdma_transfer_config sdma_transfer_config_t`

This structure configures the source/destination transfer attribute.

## 20.5.6 **typedef struct \_sdma\_buffer\_descriptor sdma\_buffer\_descriptor\_t**

This structure is a buffer descriptor, this structure describes the buffer start address and other options

## 20.5.7 **typedef struct \_sdma\_context\_data sdma\_context\_data\_t**

This structure can be load into SDMA core, with this structure, SDMA scripts can start work.

## 20.5.8 **typedef void(\* sdma\_callback)(struct \_sdma\_handle \*handle, void \*userData, bool transferDone, uint32\_t bdlIndex)**

## 20.6 Enumeration Type Documentation

### 20.6.1 **enum \_sdma\_transfer\_size**

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**

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**

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

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

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

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

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

|                           |                      |
|---------------------------|----------------------|
| <i>kSDMA_DoneChannel0</i> | SDMA done channel 0. |
| <i>kSDMA_DoneChannel1</i> | SDMA done channel 1. |
| <i>kSDMA_DoneChannel2</i> | SDMA done channel 2. |
| <i>kSDMA_DoneChannel3</i> | SDMA done channel 3. |
| <i>kSDMA_DoneChannel4</i> | SDMA done channel 4. |
| <i>kSDMA_DoneChannel5</i> | SDMA done channel 5. |
| <i>kSDMA_DoneChannel6</i> | SDMA done channel 6. |
| <i>kSDMA_DoneChannel7</i> | SDMA done channel 7. |

### 20.6.12 enum \_sdma\_done\_src

Enumerator

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kSDMA_DoneSrcSW</i>         | software done                       |
| <i>kSDMA_DoneSrcHwEvent0U</i>  | HW event 0 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent1U</i>  | HW event 1 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent2U</i>  | HW event 2 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent3U</i>  | HW event 3 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent4U</i>  | HW event 4 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent5U</i>  | HW event 5 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent6U</i>  | HW event 6 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent7U</i>  | HW event 7 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent8U</i>  | HW event 8 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent9U</i>  | HW event 9 is used for DONE event.  |
| <i>kSDMA_DoneSrcHwEvent10U</i> | HW event 10 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent11U</i> | HW event 11 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent12U</i> | HW event 12 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent13U</i> | HW event 13 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent14U</i> | HW event 14 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent15U</i> | HW event 15 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent16U</i> | HW event 16 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent17U</i> | HW event 17 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent18U</i> | HW event 18 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent19U</i> | HW event 19 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent20U</i> | HW event 20 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent21U</i> | HW event 21 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent22U</i> | HW event 22 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent23U</i> | HW event 23 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent24U</i> | HW event 24 is used for DONE event. |
| <i>kSDMA_DoneSrcHwEvent25U</i> | 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 prevent 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\_InstallIBDMemory ( *sdma\_handle\_t \* handle*, *sdma\_buffer\_descriptor\_t \* BDPool*, *uint32\_t BDCount* )

This function is called after the SDMA\_CreateHandle to use multi-buffer feature.

Parameters

|                |                                                                          |
|----------------|--------------------------------------------------------------------------|
| <i>handle</i>  | SDMA handle pointer.                                                     |
| <i>BDPool</i>  | A memory pool to store BDs. It must be located in non-cacheable address. |
| <i>BDCount</i> | The number of BD slots.                                                  |

### 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 shoule 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> | <i>fifoOffset</i> = 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\_PrepTransfer ( 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 sdma_transfer_t.                       |
| <i>srcAddr</i>          | SDMA transfer source address.                                                   |
| <i>destAddr</i>         | SDMA transfer destination address.                                              |
| <i>srcWidth</i>         | SDMA transfer source address width(bytes).                                      |
| <i>destWidth</i>        | SDMA transfer destination address width(bytes).                                 |
| <i>bytesEachRequest</i> | SDMA transfer bytes per channel request.                                        |
| <i>transferSize</i>     | SDMA transfer bytes to be transferred.                                          |
| <i>eventSource</i>      | Event source number for the transfer, if use software trigger, just write 0.    |
| <i>peripheral</i>       | Peripheral type, used to decide if need to use some special scripts.            |
| <i>type</i>             | SDMA transfer type. Used to decide the correct SDMA script address in SDMA ROM. |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error.

**20.7.32 void SDMA\_PreparesP2PTransfer ( 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 sdma_transfer_t. |
| <i>srcAddr</i>  | SDMA transfer source address.                             |
| <i>destAddr</i> | SDMA transfer destination address.                        |

|                         |                                                                      |
|-------------------------|----------------------------------------------------------------------|
| <i>srcWidth</i>         | SDMA transfer source address width(bytes).                           |
| <i>destWidth</i>        | SDMA transfer destination address width(bytes).                      |
| <i>bytesEachRequest</i> | SDMA transfer bytes per channel request.                             |
| <i>transferSize</i>     | SDMA transfer bytes to be transferred.                               |
| <i>eventSource</i>      | Event source number for the transfer.                                |
| <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 transferred 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 bool SDMA\_IsPeripheralInSPBA ( **uint32\_t addr** )

Parameters

|             |                              |
|-------------|------------------------------|
| <i>addr</i> | Address which need to judge. |
|-------------|------------------------------|

Return values

|             |                                         |
|-------------|-----------------------------------------|
| <i>True</i> | means located in SPBA, false means not. |
|-------------|-----------------------------------------|

### 20.7.39 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:  $(1 << 0) | (1 << 1)$  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\\_thresold\\_config](#)  
*configuration for TMU thresold.* [More...](#)
- struct [\\_tmu\\_interrupt\\_status](#)  
*TMU interrupt status.* [More...](#)
- struct [\\_tmu\\_config](#)  
*Configuration for TMU module.* [More...](#)

## Macros

- #define [FSL\\_TMU\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 1, 1))  
*TMU driver version.*

## Typedefs

- typedef struct [\\_tmu\\_thresold\\_config](#) [tmu\\_thresold\\_config\\_t](#)  
*configuration for TMU thresold.*
- typedef struct [\\_tmu\\_interrupt\\_status](#) [tmu\\_interrupt\\_status\\_t](#)  
*TMU interrupt status.*
- typedef enum [\\_tmu\\_average\\_low\\_pass\\_filter](#) [tmu\\_average\\_low\\_pass\\_filter\\_t](#)  
*Average low pass filter setting.*
- typedef enum [\\_tmu\\_amplifier\\_gain](#) [tmu\\_amplifier\\_gain\\_t](#)  
*Amplifier gain setting.*
- typedef enum [\\_tmu\\_amplifier\\_reference\\_voltage](#) [tmu\\_amplifier\\_reference\\_voltage\\_t](#)  
*Amplifier reference voltage setting.*
- typedef struct [\\_tmu\\_config](#) [tmu\\_config\\_t](#)  
*Configuration for TMU module.*

## 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_ATTE_MASK`,
   
    `kTMU_AverageTemperatureCriticalStatusFlags` }
   
    *TMU interrupt status flags.*
- enum `_tmu_average_low_pass_filter` {
   
    `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` {
   
    `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` {

```

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\\_SetHighTemperatureThresold](#) (TMU\_Type \*base, const [tmu\\_thresold\\_config\\_t](#) \*config)  
*Configure the high temperature thresold value and enable/disable relevant thresold.*

## 22.3 Data Structure Documentation

### 22.3.1 struct \_tmu\_thresold\_config

#### Data Fields

- bool [immediateThresoldEnable](#)  
*Enable high temperature immediate threshold.*
- bool [AverageThresoldEnable](#)  
*Enable high temperature average threshold.*
- bool [AverageCriticalThresoldEnable](#)  
*Enable high temperature average critical threshold.*
- uint8\_t [immediateThresoldValue](#)  
*Range:10U-125U.*
- uint8\_t [averageThresoldValue](#)  
*Range:10U-125U.*
- uint8\_t [averageCriticalThresoldValue](#)  
*Range:10U-125U.*

#### Field Documentation

- (1) **bool \_tmu\_thresold\_config::immediateThresoldEnable**
- (2) **bool \_tmu\_thresold\_config::AverageThresoldEnable**
- (3) **bool \_tmu\_thresold\_config::AverageCriticalThresoldEnable**
- (4) **uint8\_t \_tmu\_thresold\_config::immediateThresoldValue**

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\_thresold\_config::averageThresoldValue**

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_thresold_config::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****Data Fields**

- `uint32_t interruptDetectMask`  
*The mask of interrupt status flags.*

**Field Documentation**(1) `uint32_t _tmu_interrupt_status::interruptDetectMask`

Refer to "\_tmu\_interrupt\_status\_flags" enumeration.

**22.3.3 struct \_tmu\_config****Data Fields**

- `tmu_average_low_pass_filter_t averageLPF`  
*The average temperature is calculated as: ALPF x Current\_Temp + (1 - ALPF) x Average\_Temp.*

**Field Documentation**(1) `tmu_average_low_pass_filter_t _tmu_config::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

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

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

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`  
*Defines WDOG work mode. [More...](#)*
- struct `_wdog_config`  
*Describes WDOG configuration structure. [More...](#)*

### Typedefs

- typedef struct `_wdog_work_mode` `wdog_work_mode_t`  
*Defines WDOG work mode.*
- typedef struct `_wdog_config` `wdog_config_t`  
*Describes WDOG configuration structure.*

### Enumerations

- enum `_wdog_interrupt_enable` { `kWDOG_InterruptEnable` = `WDOG_WICR_WIE_MASK` }  
*WDOG interrupt configuration structure, default settings all disabled.*
- 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, 2, 0)`)  
*Defines WDOG driver version.*

### Refresh sequence

- #define `WDOG_REFRESH_KEY` (0xAAAA5555U)

## WDOG Initialization and De-initialization.

- void [WDOG\\_GetDefaultConfig](#) (wdog\_config\_t \*config)  
*Initializes the WDOG configuration structure.*
- void [WDOG\\_Init](#) (WDOG\_Type \*base, const wdog\_config\_t \*config)  
*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

#### Data Fields

- bool [enableWait](#)  
*If set to true, WDOG continues in wait mode.*
- bool [enableStop](#)  
*If set to true, WDOG continues in stop mode.*
- bool [enableDebug](#)  
*If set to true, WDOG continues in debug mode.*

### 23.3.2 struct \_wdog\_config

#### 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::enableTimeOutAssert**

## 23.4 Typedef Documentation

23.4.1 **typedef struct \_wdog\_work\_mode wdog\_work\_mode\_t**

23.4.2 **typedef struct \_wdog\_config wdog\_config\_t**

## 23.5 Enumeration Type Documentation

23.5.1 **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.

23.5.2 **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

## 23.6 Function Documentation

### 23.6.1 void WDOG\_GetDefaultConfig ( wdog\_config\_t \* *config* )

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
* wdogConfig->enableWdog = true;
* wdogConfig->workMode.enableWait = true;
* wdogConfig->workMode.enableStop = true;
* wdogConfig->workMode.enableDebug = true;
* wdogConfig->enableInterrupt = false;
* wdogConfig->enablePowerdown = false;
* wdogConfig->resetExtension = flase;
* wdogConfig->timeoutValue = 0xFFU;
* wdogConfig->interruptTimeValue = 0x04u;
*
```

#### Parameters

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

#### See Also

[wdog\\_config\\_t](#)

### 23.6.2 void WDOG\_Init ( WDOG\_Type \* *base*, const wdog\_config\_t \* *config* )

This function initializes the WDOG. When called, the WDOG runs according to the configuration.  
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 |
|---------------|---------------------------|

### 23.6.3 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).

### 23.6.4 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 |
|-------------|------------------------------|

### 23.6.5 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 |
|-------------|------------------------------|

### 23.6.6 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 |
|-------------|------------------------------|

### 23.6.7 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 |
|-------------|------------------------------|

### 23.6.8 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> |

### 23.6.9 **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.

### 23.6.10 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 |

### 23.6.11 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. |

### 23.6.12 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. |

### 23.6.13 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 |
|-------------|------------------------------|

### 23.6.14 void WDOG\_Refresh ( WDOG\_Type \* *base* )

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | WDOG peripheral base address |
|-------------|------------------------------|

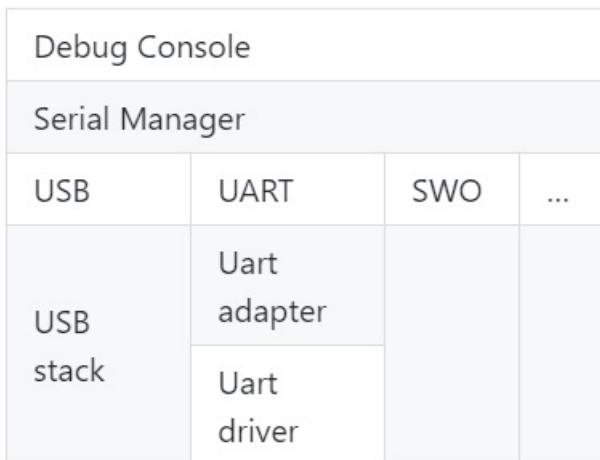
# Chapter 24

## 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.                                                          |

| .precision | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .number    | For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed. |
| .*         | The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| length         | Description |
|----------------|-------------|
| Do not support |             |

| specifier | Description                                  |
|-----------|----------------------------------------------|
| d or i    | Signed decimal integer                       |
| f         | Decimal floating point                       |
| F         | Decimal floating point capital letters       |
| x         | Unsigned hexadecimal integer                 |
| X         | Unsigned hexadecimal integer capital letters |
| o         | Signed octal                                 |
| b         | Binary value                                 |
| p         | Pointer address                              |
| u         | Unsigned decimal integer                     |
| c         | Character                                    |
| s         | String of characters                         |
| n         | Nothing printed                              |

| 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 MCUXpresso, if the macro `SDK_DEBUGCONSOLE_UART` is defined, `_sys_write` and `_sys_read` 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 show the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

| <code>SDK_DEBUGCONSOLE</code>                               | <code>SDK_DEBUGCONSOLE_UART</code> | <code>PRINTF</code>   | <code>printf</code>  |
|-------------------------------------------------------------|------------------------------------|-----------------------|----------------------|
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_SDK</code>             | defined                            | Low level peripheral* | Low level peripheral |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_SDK</code>             | undefined                          | Low level peripheral* | semihost             |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_TO-<br/>OLCHAIN</code> | defined                            | Low level peripheral* | Low level peripheral |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_TO-<br/>OLCHAIN</code> | undefined                          | semihost              | semihost             |
| <code>DEBUGCONSOLE_-<br/>DISABLE</code>                     | defined                            | No output             | Low level peripheral |
| <code>DEBUGCONSOLE_-<br/>DISABLE</code>                     | undefined                          | No output             | semihost             |

|                         |                              |               |               |
|-------------------------|------------------------------|---------------|---------------|
| <b>SDK_DEBUGCONSOLE</b> | <b>SDK_DEBUGCONSOLE_UART</b> | <b>PRINTF</b> | <b>printf</b> |
|-------------------------|------------------------------|---------------|---------------|

\* 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 equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

#### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

#### Print out failure messages using MCUXpresso SDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
 PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
 , line, func);
 for (;;) {
 }
}
```

**Note:**

To use 'printf' and 'scanf' for GNUMC 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\_B-LOCKING 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\_B-LOCKING 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>formatString-Arg</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 sued 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 ultilization, should be set per paltform'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 Complier->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 = MK64FN1M0xxxx12
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 µVision**

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

### **Step 1: Setting up the environment**

1. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log, the SDK\_DEBUGCONSOLE\_UART definition does not affect the functionality and skip the second step directly. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero, then start the second step. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one, then skip the second step directly.

**NOTE:** Case a or c only apply at example which enable swo function, the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h.

1. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
2. Open Project>Options for target or using Alt+F7 or click.
3. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
4. Select “Debug” tab and choose Port:SW, then select “Trace” tab, choose “Enable” and click OK, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

### **Step 3: Building the project**

1. Compile and link the project by choosing Project>Build Target or using F7.

### **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`  
*Initialize structure of the codec.* [More...](#)
- `struct _codec_capability`  
*codec capability* [More...](#)
- `struct _codec_handle`  
*Codec handle definition.* [More...](#)

## Macros

- `#define CODEC_VOLUME_MAX_VALUE (100U)`  
*codec maximum volume range*

## Typedefs

- `typedef enum _codec_audio_protocol codec_audio_protocol_t`  
*AUDIO format definition.*
- `typedef enum _codec_module codec_module_t`  
*audio codec module*
- `typedef enum _codec_module_ctrl_cmd codec_module_ctrl_cmd_t`  
*audio codec module control cmd*
- `typedef struct _codec_handle codec_handle_t`  
*codec handle declaration*
- `typedef struct _codec_config codec_config_t`  
*Initialize structure of the codec.*
- `typedef struct _codec_capability codec_capability_t`  
*codec capability*

## 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 {
   
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 {

```

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 { 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-initilization.*
- `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

##### Data Fields

- `uint32_t codecDevType`  
*codec type*
- `void *codecDevConfig`  
*Codec device specific configuration.*

### 25.2.2.2 struct \_codec\_capability

#### 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

- 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 Typedef Documentation

#### 25.2.4.1 typedef enum \_codec\_audio\_protocol codec\_audio\_protocol\_t

### 25.2.5 Enumeration Type Documentation

#### 25.2.5.1 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.

### 25.2.5.2 enum \_codec\_audio\_protocol

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.

### 25.2.5.3 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.

### 25.2.5.4 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

### 25.2.5.5 enum \_codec\_module

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

### 25.2.5.6 enum \_codec\_module\_ctrl\_cmd

Enumerator

*kCODEC\_ModuleSwitchI2SInInterface* module digital interface siwtch.

### 25.2.5.7 anonymous enum

Enumerator

*kCODEC\_ModuleI2SInInterfacePCM* Pcm interface.  
*kCODEC\_ModuleI2SInInterfaceDSD* DSD interface.

### 25.2.5.8 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

### 25.2.5.9 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  
*kCODEC\_RecordChannelDifferentialPositive2* differential positive record channel 2  
*kCODEC\_RecordChannelDifferentialPositive3* differential positive record channel 3  
*kCODEC\_RecordChannelDifferentialNegative1* differential negative record channel 1  
*kCODEC\_RecordChannelDifferentialNegative2* differential negative record channel 2  
*kCODEC\_RecordChannelDifferentialNegative3* differential negative record channel 3

### 25.2.5.10 anonymous enum

Enumerator

*kCODEC\_PlaySourcePGA* play source PGA, bypass ADC  
*kCODEC\_PlaySourceInput* play source Input3  
*kCODEC\_PlaySourceDAC* play source DAC  
*kCODEC\_PlaySourceMixerIn* play source mixer in  
*kCODEC\_PlaySourceMixerInLeft* play source mixer in left  
*kCODEC\_PlaySourceMixerInRight* play source mixer in right  
*kCODEC\_PlaySourceAux* play source mixer in AUX

### 25.2.5.11 anonymous enum

Enumerator

*kCODEC\_PlayChannelHeadphoneLeft* play channel headphone left  
*kCODEC\_PlayChannelHeadphoneRight* play channel headphone right  
*kCODEC\_PlayChannelSpeakerLeft* play channel speaker left  
*kCODEC\_PlayChannelSpeakerRight* play channel speaker right  
*kCODEC\_PlayChannelLineOutLeft* play channel lineout left  
*kCODEC\_PlayChannelLineOutRight* play channel lineout right  
*kCODEC\_PlayChannelLeft0* play channel left0  
*kCODEC\_PlayChannelRight0* play channel right0  
*kCODEC\_PlayChannelLeft1* play channel left1  
*kCODEC\_PlayChannelRight1* play channel right1  
*kCODEC\_PlayChannelLeft2* play channel left2  
*kCODEC\_PlayChannelRight2* play channel right2  
*kCODEC\_PlayChannelLeft3* play channel left3  
*kCODEC\_PlayChannelRight3* play channel right3

### 25.2.5.12 anonymous enum

Enumerator

*kCODEC\_VolumeHeadphoneLeft* headphone left volume

*kCODEC\_VolumeHeadphoneRight* headphone right volume  
*kCODEC\_VolumeSpeakerLeft* speaker left volume  
*kCODEC\_VolumeSpeakerRight* 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

### 25.2.5.13 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

## 25.2.6 Function Documentation

### 25.2.6.1 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.

### 25.2.6.2 status\_t CODEC\_Deinit ( *codec\_handle\_t \* handle* )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

`kStatus_Success` is success, else de-initial failed.

**25.2.6.3 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.

#### 25.2.6.4 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.

#### 25.2.6.5 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 _codec_volume_-capability or _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.

#### 25.2.6.6 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 _codec_volume_-capability or _codec_play_channel. |
| <i>mute</i>    | true is mute, false is unmute.                                                                                   |

Returns

kStatus\_Success is success, else configure failed.

#### 25.2.6.7 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.

#### 25.2.6.8 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 _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 25.2.6.9 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 _codec_record_channel, can be a value combine 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.

#### 25.2.6.10 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 _codec_play_source. |

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`  
*CODEC I2C configurations structure. [More...](#)*

## Macros

- #define `CODEC_I2C_MASTER_HANDLER_SIZE` HAL\_I2C\_MASTER\_HANDLE\_SIZE  
*codec i2c handler*

## Typedefs

- typedef enum `_codec_reg_addr` `codec_reg_addr_t`  
*CODEC device register address type.*
- typedef enum `_codec_reg_width` `codec_reg_width_t`  
*CODEC device register width.*
- typedef struct `_codec_i2c_config` `codec_i2c_config_t`  
*CODEC I2C configurations structure.*

## Enumerations

- enum `_codec_reg_addr` {
   
`kCODEC_RegAddr8Bit` = 1U,  
`kCODEC_RegAddr16Bit` = 2U }
   
*CODEC device register address type.*
- enum `_codec_reg_width` {
   
`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-initilization.*

- **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

#### Data Fields

- **uint32\_t codecI2CInstance**  
*i2c bus instance*
- **uint32\_t codecI2CSourceClock**  
*i2c bus source clock frequency*

## 25.3.3 Typedef Documentation

### 25.3.3.1 typedef enum \_codec\_reg\_addr codec\_reg\_addr\_t

### 25.3.3.2 typedef enum \_codec\_reg\_width codec\_reg\_width\_t

## 25.3.4 Enumeration Type Documentation

### 25.3.4.1 enum \_codec\_reg\_addr

Enumerator

**kCODEC\_RegAddr8Bit** 8-bit register address.

**kCODEC\_RegAddr16Bit** 16-bit register address.

### 25.3.4.2 enum \_codec\_reg\_width

Enumerator

**kCODEC\_RegWidth8Bit** 8-bit register width.

**kCODEC\_RegWidth16Bit** 16-bit register width.

**kCODEC\_RegWidth32Bit** 32-bit register width.

## 25.3.5 Function Documentation

25.3.5.1 `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.

### 25.3.5.2 status\_t CODEC\_I2C\_Deinit ( void \* *handle* )

Parameters

|               |                    |
|---------------|--------------------|
| <i>handle</i> | i2c master handle. |
|---------------|--------------------|

Returns

kStatus\_HAL\_I2cSuccess is success, else deinitial failed.

### 25.3.5.3 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.

### 25.3.5.4 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`  
*Initialize DSD mode structure of AK4497. [More...](#)*
- struct `_ak4497_pcm_config`  
*Initialize PCM mode structure of AK4497. [More...](#)*
- struct `_ak4497_config`  
*Initialize structure of AK4497. [More...](#)*
- struct `_ak4497_handle`  
*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` (100000U)  
*AK4497 i2c baudrate.*

## Typedefs

- typedef enum `_ak4497_mode` `ak4497_mode_t`  
*The AK4497 playback mode.*
- typedef enum `_ak4497_data_channel_mode` `ak4497_data_channel_mode_t`  
*The Data selection of L-channel and R-channel for DSD mode, defined by SELLR bit.*
- typedef enum `_ak4497_dsd_input_path` `ak4497_dsd_input_path_t`  
*The data path select for DSD mode.*
- typedef enum `_ak4497_dsd_mclk` `ak4497_dsd_mclk_t`  
*The MCLK select for DSD mode, defined by DCKS bit.*
- typedef enum `_ak4497_dsd_dclk` `ak4497_dsd_dclk_t`  
*The DCLK select for DSD mode, defined by DSDSEL[1:0].*
- typedef enum `_ak4497_dsd_playback_path` `ak4497_dsd_playback_path_t`  
*DSD playback path.*
- typedef enum `_ak4497_dsd_data_mute` `ak4497_dsd_data_mute_t`

- *DSD mute flag.*
- **typedef enum**  
`_ak4497_dsd_dclk_polarity` **ak4497\_dsd\_dclk\_polarity\_t**  
*DSD bclk polarity.*
- **typedef enum**  
`_ak4497_pcm_samplefreqmode` **ak4497\_pcm\_samplefreqmode\_t**  
*The sampling frequency mode for PCM and EXDF mode, defined by CR01[AFSD], CR00[ACKS].*
- **typedef enum**  
`_ak4497_pcm_samplefreqselect` **ak4497\_pcm\_samplefreqselect\_t**  
*The sampling speed select, defined by DFS[2:0].*
- **typedef enum**  
`_ak4497_pcm_sdata_format` **ak4497\_pcm\_sdata\_format\_t**  
*The audio data interface modes, defined by DIF[2:0].*
- **typedef enum** `_ak4497_pcm_tdm_mode` **ak4497\_pcm\_tdm\_mode\_t**  
*The TDM mode select, defined by TDM[1:0].*
- **typedef enum** `_ak4497_pcm_sds_select` **ak4497\_pcm\_sds\_select\_t**  
*The audio data slot selection, defined by SDS[2:0].*
- **typedef enum**  
`_ak4497_module_ctrl_cmd` **ak4497\_module\_ctrl\_cmd\_t**  
*audio codec module control cmd*
- **typedef struct** `_ak4497_dsd_config` **ak4497\_dsd\_config\_t**  
*Initialize DSD mode structure of AK4497.*
- **typedef struct** `_ak4497_pcm_config` **ak4497\_pcm\_config\_t**  
*Initialize PCM mode structure of AK4497.*
- **typedef struct** `_ak4497_config` **ak4497\_config\_t**  
*Initialize structure of AK4497.*
- **typedef struct** `_ak4497_handle` **ak4497\_handle\_t**  
*ak4497 codec handler*

## Enumerations

- **enum** `_ak4497_mode`  
*The AK4497 playback mode.*
- **enum** `_ak4497_data_channel_mode` {  
`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` {  
`kAK4497_Path0` = 0x0,  
`kAK4497_Path1` = 0x1 }  
*The data path select for DSD mode.*
- **enum** `_ak4497_dsd_mclk` {  
`kAK4497_mclk512fs` = 0x0,  
`kAK4497_mclk768fs` = 0x1 }  
*The MCLK select for DSD mode, defined by DCKS bit.*
- **enum** `_ak4497_dsd_dclk` {

```
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` {
   
kAK4497\_NormalPath = 0x0,
   
kAK4497\_VolumeBypass = 0x1 }

*DSD playback path.*

- enum `_ak4497_dsd_data_mute`
  
*DSD mute flag.*
- enum `_ak4497_dsd_dclk_polarity` {
   
kAK4497\_FallingEdge = 0x0,
   
kAK4497\_RisingEdge = 0x1 }

*DSD bclk polarity.*

- enum `_ak4497_pcm_samplefreqmode` {
   
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` {
   
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` {
   
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` {
   
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`

*The audio data slot selection, defined by SDS[2:0].*

- enum `_ak4497_module_ctrl_cmd` { 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 sampleRate, 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

### 25.4.2.2 struct \_ak4497\_pcm\_config

### 25.4.2.3 struct \_ak4497\_config

#### Data Fields

- uint8\_t `slaveAddress`  
*code device slave address*
- `codec_i2c_config_t i2cConfig`  
*i2c bus configuration*

### 25.4.2.4 struct \_ak4497\_handle

#### 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

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

Enumerator

*kAK4497\_Path0* Pin 16,17,19 used.

*kAK4497\_Path1* Pin 3,4,5 used.

#### 25.4.4.3 enum \_ak4497\_dsd\_mclk

Enumerator

*kAK4497\_mclk512fs* MCLK equals 512fs.

*kAK4497\_mclk768fs* MCLK equals 768fs.

#### 25.4.4.4 enum \_ak4497\_dsd\_dclk

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

Enumerator

*kAK4497\_NormalPath* Normal path mode.

*kAK4497\_VolumeBypass* Volume Bypass mode.

#### 25.4.4.6 enum \_ak4497\_dsd\_dclk\_polarity

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

Enumerator

*kAK4497\_ManualSettingMode* Manual setting mode.

*kAK4497\_AutoSettingMode* Auto setting mode.

*kAK4497\_FsAutoDetectMode* Auto detect mode.

#### 25.4.4.8 enum \_ak4497\_pcm\_samplefreqselect

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

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

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

Enumerator

*kAK4497\_ModuleSwitchI2SInInterface* module digital interface siwtch.

#### 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\_SetVolume ( *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)

- 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 _codec_play_source. |

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 _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.

#### 25.4.6.2.11 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 _codec_play_channel. |
| <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 _codec_record_source. |

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 _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.

**25.4.6.2.19 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 _codec_play_source. |

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 _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.

## 25.5 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*
- typedef enum `_wm8524_protocol` `wm8524_protocol_t`  
*The audio data transfer protocol.*
- typedef struct `_wm8524_handle_t` `wm8524_handle_t`  
*WM8524 handler.*

## Enumerations

- enum `_wm8524_protocol` {
   
`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_setMuteIO)(uint32_t output)`

format control io function pointer

### 25.5.4.2 `typedef enum _wm8524_protocol wm8524_protocol_t`

## 25.5.5 Enumeration Type Documentation

### 25.5.5.1 `enum _wm8524_protocol`

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-initilization.*
- `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 initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `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)`

- 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 _codec_play_source. |

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 _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.

#### 25.5.7.2.11 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 _codec_play_channel. |
| <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 _codec_record_source. |

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 _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.

**25.5.7.2.19 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 _codec_play_source. |

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 _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.

# 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](#)  
*serial manager config structure* [More...](#)
- [struct \\_serial\\_manager\\_callback\\_message](#)  
*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\\_WRITE\\_TIME\\_DELAY\\_DEFAULT\\_VALUE \(1U\)](#)

- `#define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)`  
*Set the default delay time in ms used by SerialManager\_WriteTimeDelay().*
- `#define SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY (0U)`  
*Set the default delay time in ms used by SerialManager\_ReadTimeDelay().*
- `#define SERIAL_MANAGER_WRITE_HANDLE_SIZE (44U)`  
*Enable or disable SerialManager\_Task() handle RX data available notify.*
- `#define SERIAL_MANAGER_USE_COMMON_TASK (0U)`  
*Set serial manager write handle size.*
- `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 enum _serial_port_type serial_port_type_t`  
*serial port type*
- `typedef enum _serial_manager_type serial_manager_type_t`  
*serial manager type*
- `typedef struct _serial_manager_config serial_manager_config_t`  
*serial manager config structure*
- `typedef enum _serial_manager_status serial_manager_status_t`  
*serial manager error code*
- `typedef struct _serial_manager_callback_message serial_manager_callback_message_t`  
*Callback message structure.*
- `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 {`  
 `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 {`  
 `kSerialManager_NonBlocking = 0x0U,`  
 `kSerialManager_Blocking = 0x8F41U }`  
*serial manager type*
- `enum _serial_manager_status {`  
 `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 *serialConfig)`  
*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 \*pf-Callback*)  
*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

#### 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::ringBuffer`**

Besides, the memory space cannot be free during the lifetime of the serial manager module.

**26.2.2 `struct _serial_manager_callback_message`****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

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

Enumerator

- kSerialManager\_NonBlocking* None blocking handle.
- kSerialManager\_Blocking* Blocking handle.

### 26.4.3 enum \_serial\_manager\_status

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 * serialConfig )`

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="#">S 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="#">S SERIAL_MANAGER_HANDLE_DEFINE(serialHandle)</a> ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |
| <i>serialConfig</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: <b>SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle)</b> ; or <b>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</b> |

## 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_-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.                                               |

### 26.5.8 `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_-Success</i> | Successfully received all data. |
|---------------------------------------|---------------------------------|

|                                                |                                                                      |
|------------------------------------------------|----------------------------------------------------------------------|
| <code>kStatus_SerialManager_-<br/>Busy</code>  | Previous transmission still not finished; data not all received yet. |
| <code>kStatus_SerialManager_-<br/>Error</code> | 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

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>writeHandle</code> | The serial manager module handle pointer. |
| <code>buffer</code>      | Start address of the data to write.       |
| <code>length</code>      | Length of the data to write.              |

Return values

|                                                  |                                                                  |
|--------------------------------------------------|------------------------------------------------------------------|
| <code>kStatus_SerialManager_-<br/>Success</code> | Successfully sent all data.                                      |
| <code>kStatus_SerialManager_-<br/>Busy</code>    | Previous transmission still not finished; data not all sent yet. |
| <code>kStatus_SerialManager_-<br/>Error</code>   | 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

|                         |                                                       |
|-------------------------|-------------------------------------------------------|
| <code>readHandle</code> | The serial manager module handle pointer.             |
| <code>buffer</code>     | Start address of the data to store the received data. |
| <code>length</code>     | The length of the data to be received.                |

#### Return values

|                                             |                                                                      |
|---------------------------------------------|----------------------------------------------------------------------|
| <code>kStatus_SerialManager_-Success</code> | Successfully received all data.                                      |
| <code>kStatus_SerialManager_-Busy</code>    | Previous transmission still not finished; data not all received yet. |
| <code>kStatus_SerialManager_-Error</code>   | 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

|                             |                                                       |
|-----------------------------|-------------------------------------------------------|
| <code>readHandle</code>     | The serial manager module handle pointer.             |
| <code>buffer</code>         | Start address of the data to store the received data. |
| <code>length</code>         | The length of the data to be received.                |
| <code>receivedLength</code> | Length received from the ring buffer directly.        |

Return values

|                                             |                                                                      |
|---------------------------------------------|----------------------------------------------------------------------|
| <code>kStatus_SerialManager_-Success</code> | Successfully received all data.                                      |
| <code>kStatus_SerialManager_-Busy</code>    | Previous transmission still not finished; data not all received yet. |
| <code>kStatus_SerialManager_-Error</code>   | 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

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>writeHandle</code> | The serial manager module handle pointer. |
|--------------------------|-------------------------------------------|

Return values

|                                             |                                     |
|---------------------------------------------|-------------------------------------|
| <code>kStatus_SerialManager_-Success</code> | Get successfully abort the sending. |
| <code>kStatus_SerialManager_-Error</code>   | 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.*

#### Typedefs

- typedef enum  
**\_serial\_port\_uart\_parity\_mode** **serial\_port\_uart\_parity\_mode\_t**  
*serial port uart parity mode*
- typedef enum  
**\_serial\_port\_uart\_stop\_bit\_count** **serial\_port\_uart\_stop\_bit\_count\_t**  
*serial port uart stop bit count*

#### Enumerations

- enum **\_serial\_port\_uart\_parity\_mode** {  
 kSerialManager\_UartParityDisabled = 0x0U,  
 kSerialManager\_UartParityEven = 0x2U,  
 kSerialManager\_UartParityOdd = 0x3U }  
*serial port uart parity mode*
- enum **\_serial\_port\_uart\_stop\_bit\_count** {  
 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

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

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`  
*serial port swo config struct* [More...](#)

#### Macros

- #define `SERIAL_PORT_SWO_HANDLE_SIZE` (12U)  
*serial port swo handle size*

#### Typedefs

- typedef enum  
`_serial_port_swo_protocol` `serial_port_swo_protocol_t`  
*serial port swo protocol*
- typedef struct  
`_serial_port_swo_config` `serial_port_swo_config_t`  
*serial port swo config struct*

#### Enumerations

- enum `_serial_port_swo_protocol` {  
`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`

##### 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

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,  
 kCODEC\_PCM512X,  
 kCODEC\_PCM186X }
- codec type*

### 27.1.1.2 Enumeration Type Documentation

#### 27.1.1.2.1 anonymous enum

Enumerator

|                        |          |
|------------------------|----------|
| <b>kCODEC_WM8904</b>   | wm8904   |
| <b>kCODEC_WM8960</b>   | wm8960   |
| <b>kCODEC_WM8524</b>   | wm8524   |
| <b>kCODEC_SGTL5000</b> | sgtl5000 |
| <b>kCODEC_DA7212</b>   | da7212   |
| <b>kCODEC_CS42888</b>  | CS42888. |
| <b>kCODEC_CS42448</b>  | CS42448. |
| <b>kCODEC_AK4497</b>   | AK4497.  |
| <b>kCODEC_AK4458</b>   | ak4458   |
| <b>kCODEC_TFA9XXX</b>  | tfa9xxx  |
| <b>kCODEC_TFA9896</b>  | tfa9896  |
| <b>kCODEC_WM8962</b>   | wm8962   |
| <b>kCODEC_PCM512X</b>  | pcm512x  |
| <b>kCODEC_PCM186X</b>  | pcm186x  |

**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.

