

Document Number: MCUXSDKAPIRM
Rev 2.15.000
Feb 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	19
4.2.1 struct clock_arm_pll_config_t	19
4.2.2 struct clock_usb_pll_config_t	20
4.2.3 struct clock_pll_ss_config_t	20
4.2.4 struct clock_sys_pll2_config_t	21
4.2.5 struct clock_sys_pll1_config_t	21
4.2.6 struct clock_audio_pll_config_t	21
4.2.7 struct clock_root_config_t	22
4.3 Macro Definition Documentation	22
4.3.1 FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	22
4.3.2 FSL_CLOCK_DRIVER_VERSION	23
4.3.3 LPADC_CLOCKS	23
4.3.4 AOI_CLOCKS	23
4.3.5 ASRC_CLOCKS	23
4.3.6 CMP_CLOCKS	23
4.3.7 DAC_CLOCKS	24
4.3.8 DCDC_CLOCKS	24
4.3.9 ECAT_CLOCKS	24
4.3.10 EDMA_CLOCKS	24
4.3.11 ENC_CLOCKS	24
4.3.12 EWM_CLOCKS	25
4.3.13 FLEXCAN_CLOCKS	25
4.3.14 FLEXIO_CLOCKS	25
4.3.15 FLEXSPI_CLOCKS	25
4.3.16 FLEXSPI_SLV_CLOCKS	25
4.3.17 GPC_CLOCKS	26
4.3.18 GPIO_CLOCKS	26

Section No.	Title	Page No.
4.3.19	GPT_CLOCKS	26
4.3.20	I3C_CLOCKS	26
4.3.21	IEE_CLOCKS	26
4.3.22	KPP_CLOCKS	27
4.3.23	LPI2C_CLOCKS	27
4.3.24	LPIT_CLOCKS	27
4.3.25	LPSPI_CLOCKS	27
4.3.26	LPTMR_CLOCKS	27
4.3.27	LPUART_CLOCKS	28
4.3.28	PDM_CLOCKS	28
4.3.29	MU_CLOCKS	28
4.3.30	NETC_CLOCKS	28
4.3.31	OCOTP_CLOCKS	28
4.3.32	PWM_CLOCKS	29
4.3.33	SAI_CLOCKS	29
4.3.34	SEMA42_CLOCKS	29
4.3.35	SEMC_CLOCKS	29
4.3.36	SERDES_CLOCKS	30
4.3.37	SINC_CLOCKS	30
4.3.38	SPDIF_CLOCKS	30
4.3.39	SRC_CLOCKS	30
4.3.40	TMR_CLOCKS	30
4.3.41	TPM_CLOCKS	31
4.3.42	USB_CLOCKS	31
4.3.43	USDHC_CLOCKS	31
4.3.44	WDOG_CLOCKS	31
4.3.45	XBAR_CLOCKS	31
4.3.46	CLOCK_GetCoreSysClkFreq	32
4.4	Enumeration Type Documentation	32
4.4.1	clock_lpcg_t	32
4.4.2	clock_name_t	35
4.4.3	clock_root_t	36
4.4.4	clock_root_mux_source_t	38
4.4.5	clock_osc_t	44
4.4.6	clock_gate_value_t	45
4.4.7	clock_mode_t	45
4.4.8	clock_usb_src_t	45
4.4.9	clock_usb_phy_src_t	45
4.4.10	_clock_pll_clk_src	45
4.4.11	clock_pll_post_div_t	45
4.4.12	clock_pll_t	46
4.4.13	clock_pfd_t	46
4.4.14	clock_control_mode_t	46
4.4.15	clock_24MOsc_mode_t	46

Section No.	Title	Page No.
4.4.16	<code>clock_1MHzOut_behavior_t</code>	47
4.4.17	<code>clock_level_t</code>	47
4.5	Function Documentation	47
4.5.1	<code>CLOCK_SetRootClockMux</code>	47
4.5.2	<code>CLOCK_GetRootClockMux</code>	47
4.5.3	<code>CLOCK_GetRootClockSource</code>	48
4.5.4	<code>CLOCK_SetRootClockDiv</code>	48
4.5.5	<code>CLOCK_GetRootClockDiv</code>	48
4.5.6	<code>CLOCK_PowerOffRootClock</code>	49
4.5.7	<code>CLOCK_PowerOnRootClock</code>	50
4.5.8	<code>CLOCK_SetRootClock</code>	50
4.5.9	<code>CLOCK_ControlGate</code>	50
4.5.10	<code>CLOCK_EnableClock</code>	50
4.5.11	<code>CLOCK_DisableClock</code>	51
4.5.12	<code>CLOCK_GetFreq</code>	51
4.5.13	<code>CLOCK_GetRootClockFreq</code>	51
4.5.14	<code>CLOCK_GetM7Freq</code>	52
4.5.15	<code>CLOCK_GetM33Freq</code>	52
4.5.16	<code>CLOCK_IsPllBypassed</code>	52
4.5.17	<code>CLOCK_IsPllEnabled</code>	52
4.5.18	<code>CLOCK_GetRtcFreq</code>	53
4.5.19	<code>CLOCK_SetClockSourceControlMode</code>	53
4.5.20	<code>CLOCK_OSC_EnableOscRc24M</code>	53
4.5.21	<code>CLOCK_OSC_EnableOsc24M</code>	53
4.5.22	<code>CLOCK_OSC_GateOsc24M</code>	53
4.5.23	<code>CLOCK_OSC_SetOsc24MWorkMode</code>	54
4.5.24	<code>CLOCK_OSC_EnableOscRc400M</code>	54
4.5.25	<code>CLOCK_OSC_GateOscRc400M</code>	54
4.5.26	<code>CLOCK_OSC_TrimOscRc400M</code>	54
4.5.27	<code>CLOCK_OSC_SetOscRc400MRefClkDiv</code>	55
4.5.28	<code>CLOCK_OSC_SetOscRc400MFastClkCount</code>	55
4.5.29	<code>CLOCK_OSC_SetOscRc400MHysteresisValue</code>	55
4.5.30	<code>CLOCK_OSC_BypassOscRc400MTuneLogic</code>	55
4.5.31	<code>CLOCK_OSC_EnableOscRc400MTuneLogic</code>	56
4.5.32	<code>CLOCK_OSC_FreezeOscRc400MTuneValue</code>	56
4.5.33	<code>CLOCK_OSC_SetOscRc400MTuneValue</code>	56
4.5.34	<code>CLOCK_OSC_Set1MHzOutputBehavior</code>	56
4.5.35	<code>CLOCK_OSC_SetLocked1MHzCount</code>	57
4.5.36	<code>CLOCK_OSC_CheckLocked1MHzErrorFlag</code>	57
4.5.37	<code>CLOCK_OSC_GetCurrentOscRc400MFastClockCount</code>	57
4.5.38	<code>CLOCK_OSC_GetCurrentOscRc400MTuneValue</code>	57
4.5.39	<code>CLOCK_InitArmPll</code>	57
4.5.40	<code>CLOCK_CalcArmPllFreq</code>	58
4.5.41	<code>CLOCK_InitArmPllWithFreq</code>	58

Section No.	Title	Page No.
4.5.42	CLOCK_CalcPllSpreadSpectrum	58
4.5.43	CLOCK_InitSysPll1	58
4.5.44	CLOCK_InitSysPll2	59
4.5.45	CLOCK_IsSysPll2PfdEnabled	59
4.5.46	CLOCK_InitSysPll3	59
4.5.47	CLOCK_IsSysPll3PfdEnabled	59
4.5.48	CLOCK_SetPllBypass	60
4.5.49	CLOCK_CalcAudioPllFreq	60
4.5.50	CLOCK_InitAudioPllWithFreq	60
4.5.51	CLOCK_InitAudioPll	61
4.5.52	CLOCK_GetPllFreq	61
4.5.53	CLOCK_InitPfd	61
4.5.54	CLOCK_DeinitPfd	62
4.5.55	CLOCK_GetPfdFreq	62
4.5.56	CLOCK_EnableUsbhs0Clock	62
4.5.57	CLOCK_EnableUsbhs1Clock	63
4.5.58	CLOCK_EnableUsbhs0PhyPllClock	63
4.5.59	CLOCK_DisableUsbhs0PhyPllClock	63
4.5.60	CLOCK_EnableUsbhs1PhyPllClock	64
4.5.61	CLOCK_DisableUsbhs1PhyPllClock	65
4.5.62	CLOCK_OSCPLL_LockWhiteList	65
4.5.63	CLOCK_OSCPLL_SetWhiteList	65
4.5.64	CLOCK_OSCPLL_ControlByCpuLowPowerMode	66
4.5.65	CLOCK_ROOT_LockWhiteList	66
4.5.66	CLOCK_ROOT_SetWhiteList	66
4.5.67	CLOCK_LPCG_LockWhiteList	67
4.5.68	CLOCK_LPCG_SetWhiteList	67
4.5.69	CLOCK_LPCG_ControlByCpuLowPowerMode	67

Chapter 5 DCDC_SOC driver

5.1	Overview	69
5.2	Data Structure Documentation	74
5.2.1	struct dcdc_config_t	74
5.2.2	struct dcdc_min_power_config_t	75
5.2.3	struct dcdc_detection_config_t	75
5.2.4	struct dcdc_loop_control_config_t	76
5.2.5	struct dcdc_internal_regulator_config_t	77
5.3	Macro Definition Documentation	77
5.3.1	FSL_DCDC_DRIVER_VERSION	77
5.3.2	VDD1P0_TARGET_VOLTAGE	77
5.3.3	VDD1P8_TARGET_VOLTAGE	78

Section No.	Title	Page No.
5.4 Enumeration Type Documentation		78
5.4.1 <code>_dcdc_status_flags</code>		78
5.4.2 <code>dcdc_core_slice_t</code>		78
5.4.3 <code>dcdc_control_mode_t</code>		78
5.4.4 <code>dcdc_trim_input_mode_t</code>		79
5.4.5 <code>dcdc_1P0_target_vol_t</code>		79
5.4.6 <code>dcdc_1P8_target_vol_t</code>		80
5.4.7 <code>dcdc_comparator_current_bias_t</code>		80
5.4.8 <code>dcdc_peak_current_threshold_t</code>		81
5.4.9 <code>dcdc_clock_source_t</code>		81
5.4.10 <code>dcdc_voltage_output_sel_t</code>		81
5.5 Function Documentation		81
5.5.1 <code>DCDC_Init</code>		81
5.5.2 <code>DCDC_Deinit</code>		82
5.5.3 <code>DCDC_GetDefaultConfig</code>		82
5.5.4 <code>DCDC_GetVDD1P0LowPowerModeTargetVoltage</code>		82
5.5.5 <code>DCDC_SetVDD1P0BuckModeTargetVoltage</code>		82
5.5.6 <code>DCDC_GetVDD1P0BuckModeTargetVoltage</code>		83
5.5.7 <code>DCDC_GPC_SetVDD1P0LowPowerModeTargetVoltage</code>		83
5.5.8 <code>DCDC_GPC_EnableVDD1P0LowPowerMode</code>		83
5.5.9 <code>DCDC_SetVDD1P8TargetVoltage</code>		84
5.5.10 <code>DCDC_GetVDD1P8TargetVoltage</code>		84
5.5.11 <code>DCDC_EnableVDD1P0TargetVoltageStepping</code>		84
5.5.12 <code>DCDC_GetDefaultDetectionConfig</code>		85
5.5.13 <code>DCDC_SetDetectionConfig</code>		85
5.5.14 <code>DCDC_SetClockSource</code>		85
5.5.15 <code>DCDC_SetBandgapVoltageTrimValue</code>		86
5.5.16 <code>DCDC_GetDefaultLoopControlConfig</code>		86
5.5.17 <code>DCDC_SetLoopControlConfig</code>		86
5.5.18 <code>DCDC_SetInternalRegulatorConfig</code>		87
5.5.19 <code>DCDC_EnableAdjustDelay</code>		87
5.5.20 <code>DCDC_GetStatusFlags</code>		87
5.5.21 <code>DCDC_BootIntoCCM</code>		87

Chapter 6 ROMAPI Driver

6.1 Overview		89
6.2 Data Structure Documentation		94
6.2.1 <code>struct serial_nor_config_option_t</code>		94
6.2.2 <code>struct flexspi_lut_seq_t</code>		94
6.2.3 <code>struct flexspi_dll_time_t</code>		94
6.2.4 <code>struct flexspi_mem_config_t</code>		94
6.2.5 <code>struct flexspi_nor_config_t</code>		96

Section No.	Title	Page No.
6.2.6	struct flexspi_xfer_t	97
6.3	Macro Definition Documentation	98
6.3.1	FSL_ROM_ROMAPI_VERSION	98
6.3.2	FSL_ROM_FLEXSPINOR_DRIVER_VERSION	98
6.3.3	FSL_ROM_HAS_FLEXSPINOR_API	98
6.3.4	FSL_ROM_HAS_RUNBOOTLOADER_API	98
6.3.5	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_GET_CONFIG	98
6.3.6	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_FLASH_INIT	98
6.3.7	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE	98
6.3.8	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_SECTOR	98
6.3.9	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_BLOCK	98
6.3.10	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_ALL	98
6.3.11	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_READ	98
6.3.12	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_UPDATE_LUT	98
6.3.13	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_CMD_XFER	98
6.3.14	kROM_StatusGroup_FLEXSPINOR	98
6.3.15	FLEXSPI_CFG_BLK_TAG	98
6.3.16	NOR_CMD_LUT_SEQ_IDX_READ	98
6.4	Enumeration Type Documentation	99
6.4.1	anonymous enum	99
6.4.2	anonymous enum	99
6.4.3	_flexspi_read_sample_clk	99
6.4.4	anonymous enum	99
6.4.5	anonymous enum	100
6.4.6	_flexspi_serial_clk_freq	100
6.4.7	anonymous enum	100
6.4.8	anonymous enum	100
6.4.9	_flexspi_nor_status	101
6.4.10	flexspi_operation_t	101
6.5	Function Documentation	101
6.5.1	ROM_API_Init	101
6.5.2	ROM_RunBootloader	101
6.5.3	ROM_FLEXSPI_NorFlash_GetConfig	102
6.5.4	ROM_FLEXSPI_NorFlash_Init	102
6.5.5	ROM_FLEXSPI_NorFlash_ProgramPage	103
6.5.6	ROM_FLEXSPI_NorFlash_Read	104
6.5.7	ROM_FLEXSPI_NorFlash_Erase	105
6.5.8	ROM_FLEXSPI_NorFlash_EraseSector	106
6.5.9	ROM_FLEXSPI_NorFlash_EraseBlock	107
6.5.10	ROM_FLEXSPI_NorFlash_EraseAll	108
6.5.11	ROM_FLEXSPI_NorFlash_CommandXfer	109
6.5.12	ROM_FLEXSPI_NorFlash_UpdateLut	110

Section No.	Title	Page No.
6.5.13	ROM_FLEXSPI_NorFlash_WaitBusy	110
6.5.14	ROM_FLEXSPI_NorFlash_ClearCache	111

Chapter 7 IOMUXC: IOMUX Controller

7.1	Overview	112
7.2	Macro Definition Documentation	173
7.2.1	FSL_IOMUXC_DRIVER_VERSION	173
7.3	Function Documentation	173
7.3.1	IOMUXC_SetPinMux	173
7.3.2	IOMUXC_SetPinConfig	173

Chapter 8 NIC301 Driver

Chapter 9 PMU Driver

9.1	Overview	176
9.2	Data Structure Documentation	181
9.2.1	struct pmu_static_aon_ana_ldo_config_t	181
9.2.2	struct pmu_static_aon_dig_config_t	182
9.2.3	struct pmu_bbsm_dig_config_t	183
9.2.4	struct pmu_static_bandgap_config_t	183
9.2.5	union pmu_well_bias_option_t	184
9.2.6	struct pmu_well_bias_config_t	185
9.3	Macro Definition Documentation	186
9.3.1	FSL_PMU_DRIVER_VERSION	186
9.4	Enumeration Type Documentation	186
9.4.1	pmu_ldo_name_t	186
9.4.2	pmu_control_mode_t	186
9.4.3	pmu_ldo_operate_mode_t	186
9.4.4	pmu_aon_ana_ldo_charge_pump_current_t	186
9.4.5	pmu_aon_ana_ldo_output_range_t	187
9.4.6	pmu_aon_dig_voltage_step_time_t	187
9.4.7	pmu_aon_dig_target_output_voltage_t	187
9.4.8	pmu_bbsm_dig_charge_pump_current_t	188
9.4.9	pmu_bbsm_dig_discharge_resistor_value_t	188
9.4.10	_pmu_static_bandgap_power_down_option	188
9.4.11	pmu_bandgap_output_VBG_voltage_value_t	189
9.4.12	pmu_bandgap_output_current_value_t	189
9.4.13	pmu_well_bias_power_source_t	189

Section No.	Title	Page No.
9.4.14	<code>pmu_bias_area_size_t</code>	189
9.4.15	<code>pmu_well_bias_typical_freq_t</code>	190
9.4.16	<code>pmu_adaptive_clock_source_t</code>	190
9.4.17	<code>pmu_freq_reduction_t</code>	190
9.4.18	<code>pmu_well_bias_1P8_adjustment_t</code>	190
9.5	Function Documentation	191
9.5.1	<code>PMU_SetPllLdoControlMode</code>	191
9.5.2	<code>PMU_StaticEnablePllLdo</code>	191
9.5.3	<code>PMU_StaticGetAonAnaLdoDefaultConfig</code>	191
9.5.4	<code>PMU_StaticAonAnaLdoInit</code>	192
9.5.5	<code>PMU_StaticAonAnaLdoDeinit</code>	192
9.5.6	<code>PMU_StaticGetAonDigLdoDefaultConfig</code>	192
9.5.7	<code>PMU_StaticAonDigLdoInit</code>	192
9.5.8	<code>PMU_StaticAonDigLdoDeinit</code>	193
9.5.9	<code>PMU_GetBbsmDigLdoDefaultConfig</code>	193
9.5.10	<code>PMU_EnableLdoStandbyMode</code>	193
9.5.11	<code>PMU_DisableBandgapSelfBiasAfterPowerUp</code>	193
9.5.12	<code>PMU_EnableBandgapSelfBiasBeforePowerDown</code>	194
9.5.13	<code>PMU_StaticBandgapInit</code>	194
9.5.14	<code>PMU_WellBiasInit</code>	194
9.5.15	<code>PMU_GetWellBiasDefaultConfig</code>	194
9.5.16	<code>PMU_EnableFBB</code>	194
9.5.17	<code>PMU_EnableFBBStandbyMode</code>	194
Chapter 10 SOC_SRC Driver		
10.1	Overview	195
10.2	Macro Definition Documentation	198
10.2.1	<code>FSL_SRC_DRIVER_VERSION</code>	198
10.3	Enumeration Type Documentation	199
10.3.1	<code>src_core_name_t</code>	199
10.3.2	<code>src_boot_fuse_selection_t</code>	199
10.3.3	<code>src_reset_source_t</code>	199
10.3.4	<code>_src_reset_status_flags</code>	199
10.3.5	<code>src_reset_mode_t</code>	200
10.3.6	<code>src_general_purpose_register_index_t</code>	200
10.3.7	<code>src_power_ctrl_step_t</code>	201
10.3.8	<code>src_power_level_t</code>	201
10.4	Function Documentation	201
10.4.1	<code>SRC_ReleaseCM7</code>	201
10.4.2	<code>SRC_GetBootConfig</code>	201

Section No.	Title	Page No.
10.4.3	<code>SRC_GetBootMode</code>	202
10.4.4	<code>SRC_SetGlobalSystemResetMode</code>	202
10.4.5	<code>SRC_GetResetStatusFlags</code>	202
10.4.6	<code>SRC_ClearGlobalSystemResetStatus</code>	203
10.4.7	<code>SRC_SetGeneralPurposeRegister</code>	203
10.4.8	<code>SRC_GetGeneralPurposeRegister</code>	203
10.4.9	<code>SRC_SLICE_AllowUserModeAccess</code>	204
10.4.10	<code>SRC_SLICE_AllowNonSecureModeAccess</code>	204
10.4.11	<code>SRC_SLICE_LockAccessSetting</code>	204
10.4.12	<code>SRC_SLICE_SetWhiteList</code>	205
10.4.13	<code>SRC_SLICE_LockWhiteList</code>	205
10.4.14	<code>SRC_SLICE_SoftwareControl</code>	205
10.4.15	<code>SRC_SLICE_PowerDown</code>	205

Chapter 11 ASRC: Asynchronous sample rate converter

11.1	Overview	206
11.2	ASRC Driver	207
11.2.1	Overview	207
11.2.2	Data Structure Documentation	212
11.2.3	Macro Definition Documentation	215
11.2.4	Enumeration Type Documentation	215
11.2.5	Function Documentation	218

Chapter 12 CACHE: ARMV7-M7 CACHE Memory Controller

12.1	Overview	228
12.2	Function groups	228
12.2.1	L1 CACHE Operation	228
12.2.2	L2 CACHE Operation	228
12.3	Macro Definition Documentation	229
12.3.1	<code>FSL_CACHE_DRIVER_VERSION</code>	229
12.4	Function Documentation	230
12.4.1	<code>L1CACHE_InvalidateICacheByRange</code>	230
12.4.2	<code>L1CACHE_InvalidateDCacheByRange</code>	231
12.4.3	<code>L1CACHE_CleanDCacheByRange</code>	231
12.4.4	<code>L1CACHE_CleanInvalidateDCacheByRange</code>	232
12.4.5	<code>ICACHE_InvalidateByRange</code>	233
12.4.6	<code>DCACHE_InvalidateByRange</code>	233
12.4.7	<code>DCACHE_CleanByRange</code>	234
12.4.8	<code>DCACHE_CleanInvalidateByRange</code>	235

Section No.	Title	Page No.
Chapter 13 CACHE: CACHE Memory Controller		
13.1 Overview	236	
13.2 Function groups	236	
13.2.1 CACHE Operation	236	
13.3 Macro Definition Documentation	237	
13.3.1 FSL_CACHE_DRIVER_VERSION	237	
13.3.2 XCACHE_LINESIZE_BYTE	237	
13.4 Function Documentation	237	
13.4.1 XCACHEGetInstanceByAddr	237	
13.4.2 XCACHE_EnableCache	237	
13.4.3 XCACHE_DisableCache	238	
13.4.4 XCACHE_InvalidateCache	238	
13.4.5 XCACHE_InvalidateCacheByRange	238	
13.4.6 XCACHE_CleanCache	238	
13.4.7 XCACHE_CleanCacheByRange	239	
13.4.8 XCACHE_CleanInvalidateCache	239	
13.4.9 XCACHE_CleanInvalidateCacheByRange	239	
13.4.10 ICACHE_InvalidateByRange	240	
13.4.11 DCACHE_InvalidateByRange	240	
13.4.12 DCACHE_CleanByRange	240	
13.4.13 DCACHE_CleanInvalidateByRange	241	
Chapter 14 Common Driver		
14.1 Overview	242	
14.2 Macro Definition Documentation	246	
14.2.1 FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ	246	
14.2.2 MAKE_STATUS	246	
14.2.3 MAKE_VERSION	246	
14.2.4 FSL_COMMON_DRIVER_VERSION	247	
14.2.5 DEBUG_CONSOLE_DEVICE_TYPE_NONE	247	
14.2.6 DEBUG_CONSOLE_DEVICE_TYPE_UART	247	
14.2.7 DEBUG_CONSOLE_DEVICE_TYPE_LPUART	247	
14.2.8 DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	247	
14.2.9 DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	247	
14.2.10 DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	247	
14.2.11 DEBUG_CONSOLE_DEVICE_TYPE_IUART	247	
14.2.12 DEBUG_CONSOLE_DEVICE_TYPE_VUSART	247	
14.2.13 DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	247	
14.2.14 DEBUG_CONSOLE_DEVICE_TYPE_SWO	247	

Section No.	Title	Page No.
14.2.15	DEBUG_CONSOLE_DEVICE_TYPE_QSCI	247
14.2.16	ARRAY_SIZE	247
14.2.17	SDK_SIZEALIGN	247
14.3	Typedef Documentation	247
14.3.1	status_t	247
14.4	Enumeration Type Documentation	248
14.4.1	_status_groups	248
14.4.2	anonymous enum	250
14.5	Function Documentation	251
14.5.1	SDK_Malloc	251
14.5.2	SDK_Free	251
14.5.3	SDK_DelayAtLeastUs	251
14.5.4	EnableIRQ	252
14.5.5	DisableIRQ	252
14.5.6	EnableIRQWithPriority	253
14.5.7	IRQ_SetPriority	253
14.5.8	IRQ_ClearPendingIRQ	254
14.5.9	DisableGlobalIRQ	254
14.5.10	EnableGlobalIRQ	254

Chapter 15 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

15.1	Overview	257
15.2	Typical use case	257
15.2.1	eDMA Operation	257
15.3	Data Structure Documentation	265
15.3.1	struct edma_channel_Preemption_config_t	265
15.3.2	struct edma_minor_offset_config_t	265
15.3.3	struct edma_channel_config_t	265
15.3.4	struct edma_transfer_config_t	266
15.3.5	struct edma_config_t	268
15.3.6	struct edma_handle_t	269
15.4	Macro Definition Documentation	270
15.4.1	FSL_EDMA_DRIVER_VERSION	270
15.5	Typedef Documentation	270
15.5.1	edma_tcd_t	270
15.5.2	edma_callback	270
15.6	Enumeration Type Documentation	271

Section No.	Title	Page No.
15.6.1	anonymous enum	271
15.6.2	edma_transfer_size_t	271
15.6.3	edma_modulo_t	271
15.6.4	edma_bandwidth_t	272
15.6.5	edma_channel_link_type_t	272
15.6.6	anonymous enum	272
15.6.7	anonymous enum	273
15.6.8	anonymous enum	273
15.6.9	edma_transfer_type_t	273
15.6.10	edma_channel_memory_attribute_t	273
15.6.11	edma_channel_swap_size_t	274
15.6.12	anonymous enum	274
15.6.13	edma_channel_access_type_t	275
15.6.14	edma_channel_protection_level_t	275
15.6.15	edma_channel_security_level_t	275
15.7	Function Documentation	275
15.7.1	EDMA_Init	275
15.7.2	EDMA_Deinit	276
15.7.3	EDMA_InstallTCD	277
15.7.4	EDMA_GetDefaultConfig	277
15.7.5	EDMA_InitChannel	277
15.7.6	EDMA_SetChannelMemoryAttribute	278
15.7.7	EDMA_SetChannelSignExtension	279
15.7.8	EDMA_SetChannelSwapSize	279
15.7.9	EDMA_SetChannelAccessType	279
15.7.10	EDMA_SetChannelMux	280
15.7.11	EDMA_GetChannelSystemBusInformation	280
15.7.12	EDMA_EnableChannelMasterIDReplication	280
15.7.13	EDMA_SetChannelSecurityLevel	281
15.7.14	EDMA_SetChannelProtectionLevel	282
15.7.15	EDMA_ResetChannel	282
15.7.16	EDMA_SetTransferConfig	282
15.7.17	EDMA_SetMinorOffsetConfig	283
15.7.18	EDMA_SetChannelPreemptionConfig	283
15.7.19	EDMA_SetChannelLink	284
15.7.20	EDMA_SetBandWidth	284
15.7.21	EDMA_SetModulo	285
15.7.22	EDMA_EnableAutoStopRequest	285
15.7.23	EDMA_EnableChannelInterrupts	285
15.7.24	EDMA_DisableChannelInterrupts	286
15.7.25	EDMA_SetMajorOffsetConfig	286
15.7.26	EDMA_ConfigChannelSoftwareTCD	286
15.7.27	EDMA_TcdReset	287
15.7.28	EDMA_TcdSetTransferConfig	287

Section No.	Title	Page No.
15.7.29	EDMA_TcdSetMinorOffsetConfig	288
15.7.30	EDMA_TcdSetChannelLink	288
15.7.31	EDMA_TcdSetBandWidth	289
15.7.32	EDMA_TcdSetModulo	289
15.7.33	EDMA_TcdEnableAutoStopRequest	290
15.7.34	EDMA_TcdEnableInterrupts	290
15.7.35	EDMA_TcdDisableInterrupts	290
15.7.36	EDMA_TcdSetMajorOffsetConfig	290
15.7.37	EDMA_EnableChannelRequest	291
15.7.38	EDMA_DisableChannelRequest	291
15.7.39	EDMA_TriggerChannelStart	291
15.7.40	EDMA_GetRemainingMajorLoopCount	292
15.7.41	EDMA_GetErrorStatusFlags	292
15.7.42	EDMA_GetChannelStatusFlags	293
15.7.43	EDMA_ClearChannelStatusFlags	293
15.7.44	EDMA_CreateHandle	293
15.7.45	EDMA_InstallTCDMemory	294
15.7.46	EDMA_SetCallback	294
15.7.47	EDMA_PreparesTransferConfig	295
15.7.48	EDMA_PreparesTransfer	296
15.7.49	EDMA_PreparesTransferTCD	297
15.7.50	EDMA_SubmitTransferTCD	298
15.7.51	EDMA_SubmitTransfer	299
15.7.52	EDMA_SubmitLoopTransfer	299
15.7.53	EDMA_StartTransfer	300
15.7.54	EDMA_StopTransfer	300
15.7.55	EDMA_AbortTransfer	300
15.7.56	EDMA_GetUnusedTCDNumber	301
15.7.57	EDMA_GetNextTCDAddress	301
15.7.58	EDMA_HandleIRQ	301
15.8	eDMA core Driver	303
15.9	eDMA soc Driver	304
15.9.1	Overview	304
15.9.2	Macro Definition Documentation	304
 Chapter 16 FlexCAN: Flex Controller Area Network Driver		
16.1	Overview	305
16.2	FlexCAN Driver	306
16.2.1	Overview	306
16.2.2	Typical use case	306
16.2.3	Data Structure Documentation	318

Section No.	Title	Page No.
16.2.4	Macro Definition Documentation	330
16.2.5	Enumeration Type Documentation	337
16.2.6	Function Documentation	343
 Chapter 17 FLEXSPI: Flexible Serial Peripheral Interface Driver		
17.1	Overview	373
17.2	Data Structure Documentation	378
17.2.1	struct flexspi_config_t	378
17.2.2	struct flexspi_device_config_t	381
17.2.3	struct flexspi_transfer_t	382
17.2.4	struct _flexspi_handle	383
17.3	Macro Definition Documentation	384
17.3.1	FSL_FLEXSPI_DRIVER_VERSION	384
17.3.2	FLEXSPI_LUT_SEQ	384
17.4	Typedef Documentation	384
17.4.1	flexspi_transfer_callback_t	384
17.5	Enumeration Type Documentation	384
17.5.1	anonymous enum	384
17.5.2	anonymous enum	384
17.5.3	flexspi_pad_t	385
17.5.4	flexspi_flags_t	385
17.5.5	flexspi_read_sample_clock_t	386
17.5.6	flexspi_cs_interval_cycle_unit_t	386
17.5.7	flexspi_ahb_write_wait_unit_t	386
17.5.8	flexspi_ip_error_code_t	387
17.5.9	flexspi_ahb_error_code_t	387
17.5.10	flexspi_port_t	387
17.5.11	flexspi_arb_command_source_t	388
17.5.12	flexspi_command_type_t	388
17.6	Function Documentation	388
17.6.1	FLEXSPIGetInstance	388
17.6.2	FLEXSPICheckAndClearError	388
17.6.3	FLEXSPIInit	388
17.6.4	FLEXSPIGetDefaultConfig	389
17.6.5	FLEXSPIDeinit	389
17.6.6	FLEXSPIUpdateDIIValue	389
17.6.7	FLEXSPISetFlashConfig	389
17.6.8	FLEXSPLSoftwareReset	390
17.6.9	FLEXSPIEnable	390

Section No.	Title	Page No.
17.6.10	FLEXSPI_EnableInterrupts	390
17.6.11	FLEXSPI_DisableInterrupts	390
17.6.12	FLEXSPI_EnableTxDMA	391
17.6.13	FLEXSPI_EnableRxDMA	391
17.6.14	FLEXSPI_GetTxFifoAddress	391
17.6.15	FLEXSPI_GetRxFifoAddress	391
17.6.16	FLEXSPI_ResetFifos	392
17.6.17	FLEXSPI_GetFifoCounts	392
17.6.18	FLEXSPI_GetInterruptStatusFlags	392
17.6.19	FLEXSPI_ClearInterruptStatusFlags	393
17.6.20	FLEXSPI_GetArbitratorCommandSource	393
17.6.21	FLEXSPI_GetIPCommandErrorCode	393
17.6.22	FLEXSPI_GetAHBCommandErrorCode	394
17.6.23	FLEXSPI_GetBusIdleStatus	394
17.6.24	FLEXSPI_UpdateRxSampleClock	394
17.6.25	FLEXSPI_EnableIPParallelMode	395
17.6.26	FLEXSPI_EnableAHBParallelMode	395
17.6.27	FLEXSPI_UpdateLUT	395
17.6.28	FLEXSPI_WriteData	396
17.6.29	FLEXSPI_ReadData	396
17.6.30	FLEXSPI_WriteBlocking	396
17.6.31	FLEXSPI_ReadBlocking	397
17.6.32	FLEXSPI_TransferBlocking	398
17.6.33	FLEXSPI_TransferCreateHandle	398
17.6.34	FLEXSPI_TransferNonBlocking	399
17.6.35	FLEXSPI_TransferGetCount	399
17.6.36	FLEXSPI_TransferAbort	400
17.6.37	FLEXSPI_TransferHandleIRQ	400

Chapter 18 GPT: General Purpose Timer

18.1 Overview	401
18.2 Function groups	401
18.2.1 Initialization and deinitialization	401
18.3 Typical use case	401
18.3.1 GPT interrupt example	401
18.4 Data Structure Documentation	404
18.4.1 <code>struct gpt_config_t</code>	404
18.5 Enumeration Type Documentation	405
18.5.1 <code>gpt_clock_source_t</code>	405
18.5.2 <code>gpt_input_capture_channel_t</code>	405

Section No.	Title	Page No.
18.5.3	<code>gpt_input_operation_mode_t</code>	406
18.5.4	<code>gpt_output_compare_channel_t</code>	406
18.5.5	<code>gpt_output_operation_mode_t</code>	406
18.5.6	<code>gpt_interrupt_enable_t</code>	406
18.5.7	<code>gpt_status_flag_t</code>	407
18.6	Function Documentation	407
18.6.1	<code>GPT_Init</code>	407
18.6.2	<code>GPT_Deinit</code>	407
18.6.3	<code>GPT_GetDefaultConfig</code>	407
18.6.4	<code>GPT_SoftwareReset</code>	408
18.6.5	<code>GPT_SetClockSource</code>	408
18.6.6	<code>GPT_GetClockSource</code>	408
18.6.7	<code>GPT_SetClockDivider</code>	408
18.6.8	<code>GPT_GetClockDivider</code>	409
18.6.9	<code>GPT_SetOscClockDivider</code>	409
18.6.10	<code>GPT_GetOscClockDivider</code>	409
18.6.11	<code>GPT_StartTimer</code>	409
18.6.12	<code>GPT_StopTimer</code>	410
18.6.13	<code>GPT_GetCurrentTimerCount</code>	410
18.6.14	<code>GPT_SetInputOperationMode</code>	410
18.6.15	<code>GPT_GetInputOperationMode</code>	410
18.6.16	<code>GPT_GetInputCaptureValue</code>	411
18.6.17	<code>GPT_SetOutputOperationMode</code>	411
18.6.18	<code>GPT_GetOutputOperationMode</code>	412
18.6.19	<code>GPT_SetOutputCompareValue</code>	412
18.6.20	<code>GPT_GetOutputCompareValue</code>	412
18.6.21	<code>GPT_ForceOutput</code>	413
18.6.22	<code>GPT_EnableInterrupts</code>	413
18.6.23	<code>GPT_DisableInterrupts</code>	413
18.6.24	<code>GPT_GetEnabledInterrupts</code>	413
18.6.25	<code>GPT_GetStatusFlags</code>	414
18.6.26	<code>GPT_ClearStatusFlags</code>	414
Chapter 19 I3C: I3C Driver		
19.1	Overview	415
19.2	Data Structure Documentation	416
19.2.1	<code>struct i3c_device_info_t</code>	416
19.3	Macro Definition Documentation	417
19.3.1	<code>I3C_RETRY_TIMES</code>	417
19.4	Enumeration Type Documentation	417

Section No.	Title	Page No.
19.4.1	anonymous enum	417
19.4.2	i3c_hdr_mode_t	418
19.5	I3C Master Driver	419
19.5.1	Overview	419
19.5.2	Data Structure Documentation	425
19.5.3	Typedef Documentation	429
19.5.4	Enumeration Type Documentation	429
19.5.5	Function Documentation	434
19.6	I3C Slave Driver	463
19.6.1	Overview	463
19.6.2	Data Structure Documentation	466
19.6.3	Typedef Documentation	470
19.6.4	Enumeration Type Documentation	471
19.6.5	Function Documentation	473
19.7	I3C Common Driver	485
19.7.1	Overview	485
19.7.2	Data Structure Documentation	485
19.7.3	Function Documentation	487
 Chapter 20 KPP: KeyPad Port Driver		
20.1	Overview	490
20.2	Typical use case	490
20.3	Data Structure Documentation	491
20.3.1	struct kpp_config_t	491
20.4	Macro Definition Documentation	491
20.4.1	FSL_KPP_DRIVER_VERSION	491
20.5	Enumeration Type Documentation	492
20.5.1	kpp_interrupt_enable_t	492
20.5.2	kpp_sync_operation_t	492
20.6	Function Documentation	492
20.6.1	KPP_Init	492
20.6.2	KPP_Deinit	492
20.6.3	KPP_EnableInterrupts	492
20.6.4	KPP_DisableInterrupts	493
20.6.5	KPP_GetStatusFlag	493
20.6.6	KPP_ClearStatusFlag	493
20.6.7	KPP_SetSynchronizeChain	494

Section No.	Title	Page No.
20.6.8	KPP_KeyPressScanning	495
Chapter 21 LPADC: 12-bit SAR Analog-to-Digital Converter Driver		
21.1	Overview	496
21.2	Typical use case	496
21.2.1	Polling Configuration	496
21.2.2	Interrupt Configuration	496
21.3	Data Structure Documentation	502
21.3.1	struct lpadc_config_t	502
21.3.2	struct lpadc_conv_command_config_t	504
21.3.3	struct lpadc_conv_trigger_config_t	505
21.3.4	struct lpadc_conv_result_t	506
21.3.5	struct lpadc_calibration_value_t	506
21.4	Macro Definition Documentation	507
21.4.1	FSL_LPADC_DRIVER_VERSION	507
21.4.2	LPADC_GET_ACTIVE_COMMAND_STATUS	507
21.4.3	LPADC_GET_ACTIVE_TRIGGER_STATUE	507
21.5	Enumeration Type Documentation	507
21.5.1	_lpadc_status_flags	507
21.5.2	_lpadc_interrupt_enable	507
21.5.3	_lpadc_trigger_status_flags	509
21.5.4	lpadc_sample_scale_mode_t	510
21.5.5	lpadc_sample_channel_mode_t	510
21.5.6	lpadc_hardware_average_mode_t	511
21.5.7	lpadc_sample_time_mode_t	511
21.5.8	lpadc_hardware_compare_mode_t	512
21.5.9	lpadc_conversion_resolution_mode_t	512
21.5.10	lpadc_conversion_average_mode_t	512
21.5.11	lpadc_reference_voltage_source_t	513
21.5.12	lpadc_power_level_mode_t	513
21.5.13	lpadc_offset_calibration_mode_t	513
21.5.14	lpadc_trigger_priority_policy_t	513
21.6	Function Documentation	515
21.6.1	LPADC_Init	515
21.6.2	LPADC_GetDefaultConfig	515
21.6.3	LPADC_Deinit	515
21.6.4	LPADC_Enable	516
21.6.5	LPADC_DoResetFIFO0	516
21.6.6	LPADC_DoResetFIFO1	516

Section No.	Title	Page No.
21.6.7	LPADC_DoResetConfig	516
21.6.8	LPADC_GetStatusFlags	517
21.6.9	LPADC_ClearStatusFlags	517
21.6.10	LPADC_GetTriggerStatusFlags	517
21.6.11	LPADC_ClearTriggerStatusFlags	517
21.6.12	LPADC_EnableInterrupts	518
21.6.13	LPADC_DisableInterrupts	518
21.6.14	LPADC_EnableFIFO0WatermarkDMA	518
21.6.15	LPADC_EnableFIFO1WatermarkDMA	518
21.6.16	LPADC_GetConvResultCount	519
21.6.17	LPADC_GetConvResult	519
21.6.18	LPADC_GetConvResultBlocking	519
21.6.19	LPADC_SetConvTriggerConfig	520
21.6.20	LPADC_GetDefaultConvTriggerConfig	520
21.6.21	LPADC_DoSoftwareTrigger	520
21.6.22	LPADC_SetConvCommandConfig	521
21.6.23	LPADC_GetDefaultConvCommandConfig	521
21.6.24	LPADC_SetOffset12BitValue	522
21.6.25	LPADC_SetOffset16BitValue	522
21.6.26	LPADC_EnableOffsetCalibration	522
21.6.27	LPADC_SetOffsetCalibrationMode	523
21.6.28	LPADC_DoOffsetCalibration	523
21.6.29	LPADC_DoAutoCalibration	523
21.6.30	LPADC_PrepAutoCalibration	523
21.6.31	LPADC_FinishAutoCalibration	523
21.6.32	LPADC_GetCalibrationValue	524
21.6.33	LPADC_SetCalibrationValue	524

Chapter 22 LPI2C: Low Power Inter-Integrated Circuit Driver

22.1 Overview	525
22.2 Macro Definition Documentation	526
22.2.1 FSL_LPI2C_DRIVER_VERSION	526
22.2.2 I2C_RETRY_TIMES	526
22.3 Enumeration Type Documentation	526
22.3.1 anonymous enum	526
22.4 Function Documentation	526
22.4.1 LPI2CGetInstance	526
22.5 Variable Documentation	527
22.5.1 kLpi2cIrqs	527
22.5.2 s_lpi2cMasterIrr	527

Section No.	Title	Page No.
22.5.3	s_lpi2cMasterHandle	527
22.6	LPI2C Master Driver	528
22.6.1	Overview	528
22.6.2	Data Structure Documentation	532
22.6.3	Typedef Documentation	536
22.6.4	Enumeration Type Documentation	537
22.6.5	Function Documentation	539
22.7	LPI2C Slave Driver	554
22.7.1	Overview	554
22.7.2	Data Structure Documentation	556
22.7.3	Typedef Documentation	560
22.7.4	Enumeration Type Documentation	561
22.7.5	Function Documentation	562
22.8	LPI2C Master DMA Driver	571
22.8.1	Overview	571
22.8.2	Data Structure Documentation	571
22.8.3	Typedef Documentation	573
22.8.4	Function Documentation	574
22.9	LPI2C FreeRTOS Driver	577
22.9.1	Overview	577
22.9.2	Macro Definition Documentation	577
22.9.3	Function Documentation	577
22.10	LPI2C CMSIS Driver	580
22.10.1	LPI2C CMSIS Driver	580

Chapter 23 LPIT: Low-Power Interrupt Timer

23.1	Overview	582
23.2	Function groups	582
23.2.1	Initialization and deinitialization	582
23.2.2	Timer period Operations	582
23.2.3	Start and Stop timer operations	582
23.2.4	Status	583
23.2.5	Interrupt	583
23.3	Typical use case	583
23.3.1	LPIT tick example	583
23.4	Data Structure Documentation	585
23.4.1	struct lpit_chnl_params_t	585

Section No.	Title	Page No.
23.4.2	struct lpit_config_t	586
23.5	Enumeration Type Documentation	586
23.5.1	lpit_chnl_t	586
23.5.2	lpit_timer_modes_t	586
23.5.3	lpit_trigger_select_t	587
23.5.4	lpit_trigger_source_t	587
23.5.5	lpit_interrupt_enable_t	587
23.5.6	lpit_status_flags_t	588
23.6	Function Documentation	588
23.6.1	LPIT_Init	588
23.6.2	LPIT_Deinit	588
23.6.3	LPIT_GetDefaultConfig	589
23.6.4	LPIT_SetupChannel	589
23.6.5	LPIT_EnableInterrupts	589
23.6.6	LPIT_DisableInterrupts	590
23.6.7	LPIT_GetEnabledInterrupts	590
23.6.8	LPIT_GetStatusFlags	590
23.6.9	LPIT_ClearStatusFlags	590
23.6.10	LPIT_SetTimerPeriod	591
23.6.11	LPIT_SetTimerValue	591
23.6.12	LPIT_GetCurrentTimerCount	592
23.6.13	LPIT_StartTimer	592
23.6.14	LPIT_StopTimer	592
23.6.15	LPIT_Reset	593

Chapter 24 LPSPI: Low Power Serial Peripheral Interface

24.1	Overview	594
24.2	LPSPI Peripheral driver	595
24.2.1	Overview	595
24.2.2	Function groups	595
24.2.3	Typical use case	595
24.2.4	Data Structure Documentation	602
24.2.5	Macro Definition Documentation	609
24.2.6	Typedef Documentation	610
24.2.7	Enumeration Type Documentation	611
24.2.8	Function Documentation	616
24.2.9	Variable Documentation	633
24.3	LPSPI eDMA Driver	634
24.3.1	Overview	634
24.3.2	Data Structure Documentation	635

Section No.	Title	Page No.
24.3.3	Macro Definition Documentation	640
24.3.4	Typedef Documentation	640
24.3.5	Function Documentation	641
24.4	LPSPI FreeRTOS Driver	648
24.5	LPSPI CMSIS Driver	649
24.5.1	Function groups	649
24.5.2	Typical use case	650
 Chapter 25 LPTMR: Low-Power Timer		
25.1	Overview	651
25.2	Function groups	651
25.2.1	Initialization and deinitialization	651
25.2.2	Timer period Operations	651
25.2.3	Start and Stop timer operations	651
25.2.4	Status	652
25.2.5	Interrupt	652
25.3	Typical use case	652
25.3.1	LPTMR tick example	652
25.4	Data Structure Documentation	654
25.4.1	struct lptmr_config_t	654
25.5	Enumeration Type Documentation	655
25.5.1	lptmr_pin_select_t	655
25.5.2	lptmr_pin_polarity_t	655
25.5.3	lptmr_timer_mode_t	655
25.5.4	lptmr_prescaler_glitch_value_t	655
25.5.5	lptmr_prescaler_clock_select_t	656
25.5.6	lptmr_interrupt_enable_t	656
25.5.7	lptmr_status_flags_t	656
25.6	Function Documentation	656
25.6.1	LPTMR_Init	656
25.6.2	LPTMR_Deinit	657
25.6.3	LPTMR_GetDefaultConfig	657
25.6.4	LPTMR_EnableInterrupts	657
25.6.5	LPTMR_DisableInterrupts	657
25.6.6	LPTMR_GetEnabledInterrupts	658
25.6.7	LPTMR_EnableTimerDMA	658
25.6.8	LPTMR_GetStatusFlags	658
25.6.9	LPTMR_ClearStatusFlags	659

Section No.	Title	Page No.
25.6.10	LPTMR_SetTimerPeriod	660
25.6.11	LPTMR_GetCurrentTimerCount	660
25.6.12	LPTMR_StartTimer	661
25.6.13	LPTMR_StopTimer	661

Chapter 26 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

26.1	Overview	662
26.2	LPUART Driver	663
26.2.1	Overview	663
26.2.2	Typical use case	663
26.2.3	Data Structure Documentation	668
26.2.4	Macro Definition Documentation	672
26.2.5	Typedef Documentation	672
26.2.6	Enumeration Type Documentation	672
26.2.7	Function Documentation	676
26.3	LPUART eDMA Driver	693
26.3.1	Overview	693
26.3.2	Data Structure Documentation	694
26.3.3	Macro Definition Documentation	695
26.3.4	Typedef Documentation	695
26.3.5	Function Documentation	695
26.4	LPUART FreeRTOS Driver	700
26.4.1	Overview	700
26.4.2	Data Structure Documentation	700
26.4.3	Macro Definition Documentation	701
26.4.4	Function Documentation	701
26.5	LPUART CMSIS Driver	704
26.5.1	Function groups	704

Chapter 27 MSGINTR: Message Unit

27.1	Overview	706
27.2	Typical use case	706
27.3	MSGINTR Driver	707

Chapter 28 MU: Messaging Unit

28.1	Overview	708
-------------	-----------------------	------------

Section No.	Title	Page No.
28.2 Function description		708
28.2.1 MU initialization		708
28.2.2 MU message		708
28.2.3 MU flags		709
28.2.4 Status and interrupt		709
28.2.5 MU misc functions		709
28.3 Macro Definition Documentation		712
28.3.1 FSL_MU_DRIVER_VERSION		712
28.4 Enumeration Type Documentation		712
28.4.1 _mu_status_flags		712
28.4.2 _mu_interrupt_enable		712
28.4.3 _mu_interrupt_trigger		713
28.5 Function Documentation		713
28.5.1 MU_Init		713
28.5.2 MU_Deinit		713
28.5.3 MU_SendMsgNonBlocking		713
28.5.4 MU_SendMsg		714
28.5.5 MU_ReceiveMsgNonBlocking		714
28.5.6 MU_ReceiveMsg		715
28.5.7 MU_SetFlagsNonBlocking		715
28.5.8 MU_SetFlags		716
28.5.9 MU_GetFlags		716
28.5.10 MU_GetStatusFlags		716
28.5.11 MU_GetInterruptsPending		717
28.5.12 MU_ClearStatusFlags		717
28.5.13 MU_EnableInterrupts		718
28.5.14 MU_DisableInterrupts		718
28.5.15 MU_TriggerInterrupts		719
28.5.16 MU_BootCoreB		719
28.5.17 MU_HoldCoreBReset		720
28.5.18 MU_BootOtherCore		720
28.5.19 MU_HoldOtherCoreReset		720
28.5.20 MU_ResetBothSides		721
28.5.21 MU_HardwareResetOtherCore		721
28.5.22 MU_GetOtherCorePowerMode		722

Chapter 29 NETC driver

29.1 Overview		723
29.2 Data Structure Documentation		725
29.2.1 struct netc_vlan_t		725

Section No.	Title	Page No.
29.2.2	struct netc_buffer_struct_t	725
29.2.3	struct netc_frame_struct_t	726
29.2.4	struct netc_frame_attr_t	726
29.2.5	struct netc_tx_frame_info_t	727
29.2.6	struct netc_msix_entry_t	727
29.3	Enumeration Type Documentation	728
29.3.1	anonymous enum	728
29.3.2	netc_ep_event_t	728
29.3.3	netc_ep_tx_status_t	728
29.3.4	netc_vlan_tpid_select_t	729
29.3.5	netc_msix_vector_ctrl_t	729
29.3.6	netc_tx_ext_flags_t	729
29.4	API layer	730
29.5	Hardware layer	731
29.5.1	Overview	731
29.5.2	Data Structure Documentation	733
29.5.3	Macro Definition Documentation	737
29.5.4	Enumeration Type Documentation	738
29.5.5	Hardware Station Interface(SI)	741
29.5.6	Hardware ENETC	756
29.5.7	Hardware Port	758
29.5.8	Hardware Port MAC	769
29.5.9	Hardware Switch	775
29.5.10	Hardware Common Functions	783
29.5.11	Hardware Table Access Functions	785
29.6	Abbreviation in NETC driver	865
 Chapter 30 PDM: Microphone Interface		
30.1	Overview	866
30.2	PDM Driver	867
30.2.1	Overview	867
30.2.2	Typical use case	867
30.2.3	Data Structure Documentation	875
30.2.4	Enumeration Type Documentation	878
30.2.5	Function Documentation	882

Section No.	Title	Page No.
Chapter 31 PDM EDMA Driver		
Chapter 32 QTMR: Quad Timer Driver		
32.1 Overview	899	
32.2 Data Structure Documentation	903	
32.2.1 struct qtmr_config_t	903	
32.3 Enumeration Type Documentation	903	
32.3.1 qtmr_primary_count_source_t	903	
32.3.2 qtmr_input_source_t	904	
32.3.3 qtmr_counting_mode_t	904	
32.3.4 qtmr_pwm_out_state_t	904	
32.3.5 qtmr_output_mode_t	905	
32.3.6 qtmr_input_capture_edge_t	905	
32.3.7 qtmr_reload_control_t	905	
32.3.8 qtmr_debug_action_t	905	
32.3.9 qtmr_interrupt_enable_t	906	
32.3.10 qtmr_status_flags_t	906	
32.3.11 qtmr_channel_selection_t	906	
32.3.12 qtmr_dma_enable_t	906	
32.4 Function Documentation	906	
32.4.1 QTMR_Init	906	
32.4.2 QTMR_Deinit	907	
32.4.3 QTMR_GetDefaultConfig	907	
32.4.4 QTMR_SetupPwm	907	
32.4.5 QTMR_SetupInputCapture	908	
32.4.6 QTMR_EnableInterrupts	908	
32.4.7 QTMR_DisableInterrupts	909	
32.4.8 QTMR_GetEnabledInterrupts	909	
32.4.9 QTMR_GetStatus	909	
32.4.10 QTMR_ClearStatusFlags	910	
32.4.11 QTMR_SetTimerPeriod	910	
32.4.12 QTMR_SetCompareValue	911	
32.4.13 QTMR_SetLoadValue	911	
32.4.14 QTMR_GetCurrentTimerCount	911	
32.4.15 QTMR_StartTimer	912	
32.4.16 QTMR_StopTimer	912	
32.4.17 QTMR_EnableDma	912	
32.4.18 QTMR_DisableDma	913	
32.4.19 QTMR_SetPwmOutputToIdle	913	
32.4.20 QTMR_GetPwmOutputStatus	913	
32.4.21 QTMR_GetPwmChannelStatus	914	

Section No.	Title	Page No.
32.4.22	QTMR_SetPwmClockMode	914

Chapter 33 RGPIo: Rapid General-Purpose Input/Output Driver

33.1	Overview	915
33.2	Data Structure Documentation	916
33.2.1	struct rgpio_pin_config_t	916
33.3	Macro Definition Documentation	916
33.3.1	FSL_GPIO_DRIVER_VERSION	916
33.4	Enumeration Type Documentation	916
33.4.1	rgpio_pin_direction_t	916
33.4.2	rgpio_interrupt_sel_t	916
33.4.3	rgpio_interrupt_config_t	917
33.5	RGPIo Driver	918
33.5.1	Overview	918
33.5.2	Typical use case	918
33.5.3	Function Documentation	919
33.6	FGPIO Driver	925
33.6.1	Typical use case	925

Chapter 34 RTWDOG: 32-bit Watchdog Timer

34.1	Overview	926
34.2	Typical use case	926
34.3	Data Structure Documentation	928
34.3.1	struct rtwdog_work_mode_t	928
34.3.2	struct rtwdog_config_t	928
34.4	Macro Definition Documentation	929
34.4.1	FSL_RTWDOG_DRIVER_VERSION	929
34.5	Enumeration Type Documentation	929
34.5.1	rtwdog_clock_source_t	929
34.5.2	rtwdog_clock_prescaler_t	929
34.5.3	rtwdog_test_mode_t	929
34.5.4	_rtwdog_interrupt_enable_t	929
34.5.5	_rtwdog_status_flags_t	930
34.6	Function Documentation	930

Section No.	Title	Page No.
34.6.1	RTWDOG_GetDefaultConfig	930
34.6.2	RTWDOG_Init	930
34.6.3	RTWDOG_Deinit	931
34.6.4	RTWDOG_Enable	931
34.6.5	RTWDOG_Disable	931
34.6.6	RTWDOG_EnableInterrupts	932
34.6.7	RTWDOG_DisableInterrupts	932
34.6.8	RTWDOG_GetStatusFlags	932
34.6.9	RTWDOG_EnableWindowMode	933
34.6.10	RTWDOG_CountToMesec	933
34.6.11	RTWDOG_ClearStatusFlags	933
34.6.12	RTWDOG_SetTimeoutValue	934
34.6.13	RTWDOG_SetWindowValue	934
34.6.14	RTWDOG_Unlock	934
34.6.15	RTWDOG_Refresh	935
34.6.16	RTWDOG_GetCounterValue	935

Chapter 35 S3MU: Sentinel

Chapter 36 SAI: Serial Audio Interface

36.1	Overview	937
36.2	Typical configurations	937
36.3	Typical use case	938
36.3.1	SAI Send/receive using an interrupt method	938
36.3.2	SAI Send/receive using a DMA method	938
36.4	SAI Driver	939
36.4.1	Overview	939
36.4.2	Data Structure Documentation	946
36.4.3	Macro Definition Documentation	950
36.4.4	Enumeration Type Documentation	950
36.4.5	Function Documentation	955
36.5	SAI EDMA Driver	979
36.5.1	Overview	979
36.5.2	Data Structure Documentation	980
36.5.3	Enumeration Type Documentation	981
36.5.4	Function Documentation	982

Chapter 37 SEMC: Smart External DRAM Controller Driver

37.1	Overview	991
-------------	-----------------------	------------

Section No.	Title	Page No.
37.2 SEMC: Smart External DRAM Controller Driver		991
37.2.1 SEMC Initialization Operation		991
37.2.2 SEMC Interrupt Operation		991
37.2.3 SEMC Memory access Operation		991
37.3 Typical use case		991
37.4 Data Structure Documentation		998
37.4.1 struct semc_sdram_config_t		998
37.4.2 struct semc_nand_timing_config_t		1001
37.4.3 struct semc_nand_config_t		1003
37.4.4 struct semc_nor_config_t		1004
37.4.5 struct semc_sram_config_t		1007
37.4.6 struct semc_dbi_config_t		1011
37.4.7 struct semc_queuea_weight_struct_t		1013
37.4.8 union semc_queuea_weight_t		1014
37.4.9 struct semc_queueb_weight_struct_t		1014
37.4.10 union semc_queueb_weight_t		1015
37.4.11 struct semc_axi_queueweight_t		1015
37.4.12 struct semc_config_t		1016
37.5 Macro Definition Documentation		1016
37.5.1 FSL_SEMC_DRIVER_VERSION		1016
37.6 Enumeration Type Documentation		1016
37.6.1 anonymous enum		1016
37.6.2 semc_mem_type_t		1017
37.6.3 semc_waitready_polarity_t		1017
37.6.4 semc_sdram_cs_t		1017
37.6.5 semc_sram_cs_t		1017
37.6.6 semc_nand_access_type_t		1018
37.6.7 semc_interrupt_enable_t		1018
37.6.8 semc_ipcmd_datasize_t		1018
37.6.9 semc_refresh_time_t		1018
37.6.10 semc_caslatency_t		1019
37.6.11 semc_sdram_column_bit_num_t		1019
37.6.12 sem_sdram_burst_len_t		1019
37.6.13 semc_nand_column_bit_num_t		1019
37.6.14 sem_nand_burst_len_t		1020
37.6.15 semc_norsram_column_bit_num_t		1020
37.6.16 sem_norsram_burst_len_t		1020
37.6.17 semc_dbi_column_bit_num_t		1021
37.6.18 sem_dbi_burst_len_t		1021
37.6.19 semc_iomux_pin		1021
37.6.20 semc_iomux_nora27_pin		1022

Section No.	Title	Page No.
37.6.21	smec_port_size_t	1022
37.6.22	smec_addr_mode_t	1022
37.6.23	smec_dqs_mode_t	1022
37.6.24	smec_adv_polarity_t	1022
37.6.25	smec_sync_mode_t	1023
37.6.26	smec_adv_level_control_t	1023
37.6.27	smec_rdy_polarity_t	1023
37.6.28	semc_ipcmd_nand_addrmode_t	1023
37.6.29	semc_ipcmd_nand_cmdmode_t	1023
37.6.30	semc_nand_address_option_t	1024
37.6.31	semc_ipcmd_nor_dbt_t	1024
37.6.32	semc_ipcmd_sram_t	1024
37.6.33	semc_ipcmd_sdram_t	1024
37.7	Function Documentation	1025
37.7.1	SEMC_GetDefaultConfig	1025
37.7.2	SEMC_Init	1025
37.7.3	SEMC_Deinit	1025
37.7.4	SEMC_ConfigureSDRAM	1026
37.7.5	SEMC_ConfigureNAND	1026
37.7.6	SEMC_ConfigureNOR	1026
37.7.7	SEMC_ConfigureSRAMWithChipSelection	1026
37.7.8	SEMC_ConfigureSRAM	1027
37.7.9	SEMC_ConfigureDBI	1027
37.7.10	SEMC_EnableInterrupts	1027
37.7.11	SEMC_DisableInterrupts	1028
37.7.12	SEMC_GetStatusFlag	1028
37.7.13	SEMC_ClearStatusFlags	1028
37.7.14	SEMC_IsInIdle	1029
37.7.15	SEMC_SendIPCommand	1029
37.7.16	SEMC_BuildNandIPCommand	1029
37.7.17	SEMC_IsNandReady	1030
37.7.18	SEMC_IPCommandNandWrite	1030
37.7.19	SEMC_IPCommandNandRead	1030
37.7.20	SEMC_IPCommandNorWrite	1031
37.7.21	SEMC_IPCommandNorRead	1032
Chapter 38	SINC: SINC Filter	
38.1	Overview	1033
38.2	Data Structure Documentation	1043
38.2.1	struct sinc_channel_input_option_t	1043
38.2.2	struct sinc_channel_conv_option_t	1044
38.2.3	struct sinc_channel_protection_option_t	1045

Section No.	Title	Page No.
38.2.4	struct sinc_channel_config_t	1046
38.2.5	struct sinc_config_t	1047
38.3	Macro Definition Documentation	1048
38.3.1	FSL_SINC_DRIVER_VERSION	1048
38.4	Enumeration Type Documentation	1048
38.4.1	_sinc_interrupt_enable	1048
38.4.2	_sinc_interrupt_status	1048
38.4.3	sinc_channel_id_t	1048
38.4.4	_sinc_modulator_clock	1048
38.4.5	sinc_inputClk_source_t	1049
38.4.6	sinc_inputClk_edge_t	1049
38.4.7	sinc_inputBit_format_t	1049
38.4.8	sinc_inputBit_source_t	1049
38.4.9	sinc_conv_trigger_source_t	1050
38.4.10	sinc_conv_mode_t	1050
38.4.11	sinc_pulse_trigger_mux_t	1050
38.4.12	sinc_zero_cross_operate_mode_t	1051
38.4.13	sinc_primary_filter_order_t	1052
38.4.14	sinc_clock_prescale_t	1052
38.4.15	sinc_primary_filter_shift_direction_t	1052
38.4.16	sinc_primary_filter_bias_sign_t	1052
38.4.17	sin_primary_filter_hpf_alpha_coeff_t	1052
38.4.18	sinc_inputBit_delay_t	1053
38.4.19	sinc_scd_operate_mode_t	1053
38.4.20	sinc_scd_option_t	1054
38.4.21	sinc_limit_detector_mode_t	1054
38.4.22	sinc_cad_threshold_t	1054
38.4.23	sinc_result_data_format_t	1055
38.4.24	sinc_debug_output_t	1055
38.5	Function Documentation	1055
38.5.1	SINC_Init	1055
38.5.2	SINC_Deinit	1056
38.5.3	SINC_GetDefaultConfig	1056
38.5.4	SINC_EnableMaster	1056
38.5.5	SINC_DoSoftwareReset	1056
38.5.6	SINC_DisableDozeMode	1057
38.5.7	SINC_CheckModulatorClockReady	1057
38.5.8	SINC_DisableModulatorClockOutput	1057
38.5.9	SINC_SetClkPrescale	1058
38.5.10	SINC_SetModulatorClockDivider	1058
38.5.11	SINC_SetChannelConfig	1058
38.5.12	SINC_SetChannelInputOption	1059

Section No.	Title	Page No.
38.5.13	SINC_SetChannelConversionOption	1059
38.5.14	SINC_SetChannelProtectionOption	1059
38.5.15	SINC_ReadChannelResultData	1060
38.5.16	SINC_EnableChannelFIFO	1061
38.5.17	SINC_SetChannelFifoWatermark	1061
38.5.18	SINC_EnableChannel	1061
38.5.19	SINC_EnableChannelPrimaryDma	1062
38.5.20	SINC_SetChannelResultDataFormat	1062
38.5.21	SINC_GetChannelFifoCount	1062
38.5.22	SINC_CheckChannelResultDataReady	1063
38.5.23	SINC_CheckChannelFifoEmpty	1063
38.5.24	SINC_AffirmChannelSoftwareTrigger	1063
38.5.25	SINC_NegateChannelSoftwareTrigger	1064
38.5.26	SINC_SetChannelConversionMode	1064
38.5.27	SINC_SetChannelTriggerSource	1064
38.5.28	SINC_SetChannelMultipurposeData	1065
38.5.29	SINC_SetChannelPfOrder	1065
38.5.30	SINC_SetChannelPfOsr	1065
38.5.31	SINC_SetChannelPfHpfAlphaCoeff	1066
38.5.32	SINC_SetChannelPfShiftConfig	1066
38.5.33	SINC_SetChannelPfBiasConfig	1066
38.5.34	SINC_EnableChannelPrimaryFilter	1067
38.5.35	SINC_CheckChannelParallelSerialDataReady	1067
38.5.36	SINC_CheckChannelPrimaryCICSaturation	1068
38.5.37	SINC_CheckChannelHPFSaturation	1069
38.5.38	SINC_CheckChannelShiftSaturation	1069
38.5.39	SINC_CheckChannelBiasSaturation	1070
38.5.40	SINC_GetChannelConversionCount	1071
38.5.41	SINC_CheckChannelConvProgress	1071
38.5.42	SINC_CheckChannelReadyForConv	1071
38.5.43	SINC_SetChannelLowLimitThreshold	1072
38.5.44	SINC_SetChannelHighLimitThreshold	1072
38.5.45	SINC_SetChannelLimitDetectorMode	1073
38.5.46	SINC_EnableChannelHighLimitBreakSignal	1073
38.5.47	SINC_EnableChannelWindowLimitBreakSignal	1073
38.5.48	SINC_EnableChannelLowLimitBreakSignal	1074
38.5.49	SINC_SetChannelScdOperateMode	1074
38.5.50	SINC_SetChannelScdLimitThreshold	1074
38.5.51	SINC_SetChannelScdOption	1075
38.5.52	SINC_EnableChannelScdBreakSignal	1075
38.5.53	SINC_SetChannelCadLimitThreshold	1076
38.5.54	SINC_EnableChannelCadBreakSignal	1076
38.5.55	SINC_SetChannelZcdOperateMode	1076
38.5.56	SINC_SetChannelPulseTriggerMux	1077
38.5.57	SINC_SetChannelDebugOutput	1078

Section No.	Title	Page No.
38.5.58	SINC_LatchChannelDebugProcedure	1078
38.5.59	SINC_CheckChannelDebugDataValid	1078
38.5.60	SINC_GetChannelDebugData	1079
38.5.61	SINC_EnableInterrupts	1079
38.5.62	SINC_DisableInterrupts	1079
38.5.63	SINC_GetInterruptStatus	1079
38.5.64	SINC_ClearInterruptStatus	1080

Chapter 39 TEMPSENSOR: Temperature Sensor Module

39.1	Overview	1081
39.2	TEMPSENSOR Operations	1081
39.3	Data Structure Documentation	1082
39.3.1	struct tmpsns_config_t	1082
39.4	Enumeration Type Documentation	1083
39.4.1	anonymous enum	1083
39.4.2	anonymous enum	1083
39.4.3	tmpsns_measure_mode_t	1083
39.4.4	tmpsns_alarm_mode_t	1084
39.5	Function Documentation	1084
39.5.1	TMPNSN_Init	1084
39.5.2	TMPNSN_Deinit	1084
39.5.3	TMPNSN_GetDefaultConfig	1084
39.5.4	TMPNSN_StartMeasure	1084
39.5.5	TMPNSN_StopMeasure	1085
39.5.6	TMPNSN_GetCurrentTemperature	1085
39.5.7	TMPNSN_SetTempAlarm	1085
39.5.8	TMPNSN_EnableInterrupt	1085
39.5.9	TMPNSN_DisableInterrupt	1086
39.5.10	TMPNSN_GetInterruptFlags	1086
39.5.11	TMPNSN_ClearInterruptFlags	1086

Chapter 40 TPM: Timer PWM Module

40.1	Overview	1087
40.2	Introduction of TPM	1087
40.2.1	Initialization and deinitialization	1087
40.2.2	PWM Operations	1087
40.2.3	Input capture operations	1088
40.2.4	Output compare operations	1088

Section No.	Title	Page No.
40.2.5	Quad decode	1088
40.2.6	Fault operation	1088
40.2.7	Status	1088
40.2.8	Interrupt	1088
40.3	Typical use case	1089
40.3.1	PWM output	1089
40.4	Data Structure Documentation	1093
40.4.1	struct tpm_chnl_pwm_signal_param_t	1093
40.4.2	struct tpm_dual_edge_capture_param_t	1094
40.4.3	struct tpm_phase_params_t	1095
40.4.4	struct tpm_config_t	1095
40.5	Macro Definition Documentation	1096
40.5.1	FSL TPM DRIVER VERSION	1096
40.6	Enumeration Type Documentation	1096
40.6.1	tpm_chnl_t	1096
40.6.2	tpm_pwm_mode_t	1096
40.6.3	tpm_pwm_level_select_t	1096
40.6.4	tpm_pwm_pause_level_select_t	1097
40.6.5	tpm_chnl_control_bit_mask_t	1097
40.6.6	tpm_trigger_select_t	1097
40.6.7	tpm_trigger_source_t	1097
40.6.8	tpm_ext_trigger_polarity_t	1098
40.6.9	tpm_output_compare_mode_t	1098
40.6.10	tpm_input_capture_edge_t	1098
40.6.11	tpm_quad_decode_mode_t	1099
40.6.12	tpm_phase_polarity_t	1099
40.6.13	tpm_clock_source_t	1099
40.6.14	tpm_clock_prescale_t	1099
40.6.15	tpm_interrupt_enable_t	1100
40.6.16	tpm_status_flags_t	1100
40.7	Function Documentation	1100
40.7.1	TPM_Init	1100
40.7.2	TPM_Deinit	1101
40.7.3	TPM_GetDefaultConfig	1101
40.7.4	TPM_CalculateCounterClkDiv	1101
40.7.5	TPM_SetupPwm	1102
40.7.6	TPM_UpdatePwmDutyCycle	1102
40.7.7	TPM_UpdateChnlEdgeLevelSelect	1103
40.7.8	TPM_GetChannelControlBits	1103
40.7.9	TPM_DisableChannel	1104

Section No.	Title	Page No.
40.7.10	TPM_EnableChannel	1104
40.7.11	TPM_SetupInputCapture	1104
40.7.12	TPM_SetupOutputCompare	1105
40.7.13	TPM_SetupDualEdgeCapture	1105
40.7.14	TPM_SetupQuadDecode	1106
40.7.15	TPM_SetChannelPolarity	1106
40.7.16	TPM_EnableChannelExtTrigger	1106
40.7.17	TPM_EnableInterrupts	1107
40.7.18	TPM_DisableInterrupts	1107
40.7.19	TPM_GetEnabledInterrupts	1107
40.7.20	TPM_GetChannelValue	1107
40.7.21	TPM_GetStatusFlags	1108
40.7.22	TPM_ClearStatusFlags	1108
40.7.23	TPM_SetTimerPeriod	1108
40.7.24	TPM_GetCurrentTimerCount	1109
40.7.25	TPM_StartTimer	1109
40.7.26	TPM_StopTimer	1109
40.7.27	TPM_Reset	1110

Chapter 41 TRDC: Trusted Resource Domain Controller

41.1	Overview	1111
41.2	TRDC functions	1111
41.3	Data Structure Documentation	1118
41.3.1	struct trdc_hardware_config_t	1118
41.3.2	struct trdc_slave_memory_hardware_config_t	1119
41.3.3	struct trdc_processor_domain_assignment_t	1119
41.3.4	struct trdc_non_processor_domain_assignment_t	1120
41.3.5	struct trdc_pid_config_t	1121
41.3.6	struct trdc_idau_config_t	1121
41.3.7	struct trdc_flw_config_t	1122
41.3.8	struct trdc_domain_error_t	1123
41.3.9	struct trdc_memory_access_control_config_t	1124
41.3.10	struct trdc_mrc_region_descriptor_config_t	1125
41.3.11	struct trdc_mbc_nse_update_config_t	1126
41.3.12	struct trdc_mbc_memory_block_config_t	1127
41.4	Enumeration Type Documentation	1128
41.4.1	trdc_did_sel_t	1128
41.4.2	trdc_secure_attr_t	1128
41.4.3	trdc_pid_domain_hit_config_t	1128
41.4.4	trdc_privilege_attr_t	1129
41.4.5	trdc_pid_lock_t	1129

Section No.	Title	Page No.
41.4.6	<code>trdc_controller_t</code>	1129
41.4.7	<code>trdc_error_state_t</code>	1129
41.4.8	<code>trdc_error_attr_t</code>	1130
41.4.9	<code>trdc_error_type_t</code>	1130
41.4.10	<code>_trdc_region_descriptor</code>	1130
41.4.11	<code>_trdc_MRC_domain</code>	1131
41.4.12	<code>_trdc_MBC_domain</code>	1131
41.4.13	<code>_trdc_MBC_memory</code>	1131
41.4.14	<code>_trdc_MBC_bit</code>	1132
41.5	Function Documentation	1133
41.5.1	<code>TRDC_Init</code>	1133
41.5.2	<code>TRDC_Deinit</code>	1134
41.5.3	<code>TRDC_GetCurrentMasterDomainId</code>	1134
41.5.4	<code>TRDC_GetHardwareConfig</code>	1134
41.5.5	<code>TRDC_SetDacGlobalValid</code>	1134
41.5.6	<code>TRDC_LockMasterDomainAssignment</code>	1135
41.5.7	<code>TRDC_SetMasterDomainAssignmentValid</code>	1135
41.5.8	<code>TRDC_GetDefaultProcessorDomainAssignment</code>	1135
41.5.9	<code>TRDC_GetDefaultNonProcessorDomainAssignment</code>	1136
41.5.10	<code>TRDC_SetPid</code>	1136
41.5.11	<code>TRDC_GetActiveMasterPidMap</code>	1136
41.5.12	<code>TRDC_SetProcessorDomainAssignment</code>	1137
41.5.13	<code>TRDC_EnableProcessorDomainAssignment</code>	1137
41.5.14	<code>TRDC_SetNonProcessorDomainAssignment</code>	1138
41.5.15	<code>TRDC_GetDefaultIDAUConfig</code>	1138
41.5.16	<code>TRDC_SetIDAU</code>	1139
41.5.17	<code>TRDC_EnableFlashLogicalWindow</code>	1139
41.5.18	<code>TRDC_LockFlashLogicalWindow</code>	1139
41.5.19	<code>TRDC_GetFlashLogicalWindowPbase</code>	1140
41.5.20	<code>TRDC_GetSetFlashLogicalWindowSize</code>	1140
41.5.21	<code>TRDC_GetDefaultFlashLogicalWindowConfig</code>	1140
41.5.22	<code>TRDC_SetFlashLogicalWindow</code>	1141
41.5.23	<code>TRDC_GetAndClearFirstDomainError</code>	1141
41.5.24	<code>TRDC_GetAndClearFirstSpecificDomainError</code>	1141
41.5.25	<code>TRDC_SetMrcGlobalValid</code>	1142
41.5.26	<code>TRDC_GetMrcRegionNumber</code>	1142
41.5.27	<code>TRDC_MrcSetMemoryAccessConfig</code>	1142
41.5.28	<code>TRDC_MrcEnableDomainNseUpdate</code>	1143
41.5.29	<code>TRDC_MrcRegionNseSet</code>	1143
41.5.30	<code>TRDC_MrcRegionNseClear</code>	1144
41.5.31	<code>TRDC_MrcDomainNseClear</code>	1144
41.5.32	<code>TRDC_MrcSetRegionDescriptorConfig</code>	1144
41.5.33	<code>TRDC_SetMbcGlobalValid</code>	1145
41.5.34	<code>TRDC_GetMbcHardwareConfig</code>	1145

Section No.	Title	Page No.
41.5.35	TRDC_MbcSetNseUpdateConfig	1145
41.5.36	TRDC_MbcWordNseSet	1146
41.5.37	TRDC_MbcWordNseClear	1147
41.5.38	TRDC_MbcNseClearAll	1147
41.5.39	TRDC_MbcSetMemoryAccessConfig	1147
41.5.40	TRDC_MbcSetMemoryBlockConfig	1148

Chapter 42 USDHC: Ultra Secured Digital Host Controller Driver

42.1	Overview	1149
42.2	Typical use case	1149
42.2.1	USDHC Operation	1149
42.3	Data Structure Documentation	1160
42.3.1	struct usdhc_adma2_descriptor_t	1160
42.3.2	struct usdhc_capability_t	1160
42.3.3	struct usdhc_boot_config_t	1161
42.3.4	struct usdhc_config_t	1162
42.3.5	struct usdhc_command_t	1162
42.3.6	struct usdhc_adma_config_t	1163
42.3.7	struct usdhc_scatter_gather_data_list_t	1163
42.3.8	struct usdhc_scatter_gather_data_t	1164
42.3.9	struct usdhc_scatter_gather_transfer_t	1164
42.3.10	struct usdhc_data_t	1165
42.3.11	struct usdhc_transfer_t	1166
42.3.12	struct usdhc_transfer_callback_t	1166
42.3.13	struct _usdhc_handle	1167
42.3.14	struct usdhc_host_t	1168
42.4	Macro Definition Documentation	1168
42.4.1	FSL_USDHC_DRIVER_VERSION	1168
42.4.2	USDHC_ADMA1_ADDRESS_ALIGN	1168
42.4.3	USDHC_ADMA1_LENGTH_ALIGN	1168
42.4.4	USDHC_ADMA2_ADDRESS_ALIGN	1168
42.4.5	USDHC_ADMA2_LENGTH_ALIGN	1168
42.4.6	USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT	1168
42.4.7	USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK	1169
42.4.8	USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT	1169
42.4.9	USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK	1169
42.4.10	USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY	1169
42.4.11	USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT	1169
42.4.12	USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK	1169
42.4.13	USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY	1169

Section No.	Title	Page No.
42.5	Typedef Documentation	1169
42.5.1	usdhc_adma1_descriptor_t	1169
42.5.2	usdhc_transfer_function_t	1169
42.6	Enumeration Type Documentation	1170
42.6.1	anonymous enum	1170
42.6.2	anonymous enum	1170
42.6.3	anonymous enum	1171
42.6.4	anonymous enum	1171
42.6.5	anonymous enum	1171
42.6.6	anonymous enum	1172
42.6.7	anonymous enum	1172
42.6.8	anonymous enum	1173
42.6.9	anonymous enum	1174
42.6.10	anonymous enum	1174
42.6.11	anonymous enum	1174
42.6.12	anonymous enum	1174
42.6.13	usdhc_transfer_direction_t	1175
42.6.14	usdhc_data_bus_width_t	1175
42.6.15	usdhc_endian_mode_t	1175
42.6.16	usdhc_dma_mode_t	1176
42.6.17	anonymous enum	1176
42.6.18	usdhc_boot_mode_t	1176
42.6.19	usdhc_card_command_type_t	1176
42.6.20	usdhc_card_response_type_t	1176
42.6.21	anonymous enum	1177
42.6.22	anonymous enum	1177
42.6.23	anonymous enum	1178
42.6.24	usdhc_burst_len_t	1178
42.6.25	anonymous enum	1178
42.7	Function Documentation	1178
42.7.1	USDHC_Init	1178
42.7.2	USDHC_Deinit	1179
42.7.3	USDHC_Reset	1179
42.7.4	USDHC_SetAdmaTableConfig	1179
42.7.5	USDHC_SetInternalDmaConfig	1180
42.7.6	USDHC_SetADMA2Descriptor	1180
42.7.7	USDHC_SetADMA1Descriptor	1181
42.7.8	USDHC_EnableInternalDMA	1181
42.7.9	USDHC_EnableInterruptStatus	1182
42.7.10	USDHC_DisableInterruptStatus	1182
42.7.11	USDHC_EnableInterruptSignal	1182
42.7.12	USDHC_DisableInterruptSignal	1182
42.7.13	USDHC_GetEnabledInterruptStatusFlags	1183

Section No.	Title	Page No.
42.7.14	USDHC_GetInterruptStatusFlags	1183
42.7.15	USDHC_ClearInterruptStatusFlags	1183
42.7.16	USDHC_GetAutoCommand12ErrorStatusFlags	1184
42.7.17	USDHC_GetAdmaErrorStatusFlags	1185
42.7.18	USDHC_GetPresentStatusFlags	1185
42.7.19	USDHC_GetCapability	1185
42.7.20	USDHC_ForceClockOn	1186
42.7.21	USDHC_SetSdClock	1186
42.7.22	USDHC_SetCardActive	1186
42.7.23	USDHC_AssertHardwareReset	1187
42.7.24	USDHC_SetDataBusWidth	1187
42.7.25	USDHC_WriteData	1187
42.7.26	USDHC_ReadData	1187
42.7.27	USDHC_SendCommand	1188
42.7.28	USDHC_EnableWakeupEvent	1188
42.7.29	USDHC_CardDetectByData3	1188
42.7.30	USDHC_DetectCardInsert	1188
42.7.31	USDHC_EnableSdioControl	1189
42.7.32	USDHC_SetContinueRequest	1189
42.7.33	USDHC_RequestStopAtBlockGap	1189
42.7.34	USDHC_SetMmcBootConfig	1189
42.7.35	USDHC_EnableMmcBoot	1190
42.7.36	USDHC_SetForceEvent	1190
42.7.37	UDSHC_SelectVoltage	1190
42.7.38	USDHC_RequestTuningForSDR50	1190
42.7.39	USDHC_RequestReTuning	1191
42.7.40	USDHC_EnableAutoTuning	1191
42.7.41	USDHC_EnableAutoTuningForCmdAndData	1191
42.7.42	USDHC_EnableManualTuning	1191
42.7.43	USDHC_GetTuningDelayStatus	1192
42.7.44	USDHC_SetTuningDelay	1192
42.7.45	USDHC_AdjustDelayForManualTuning	1192
42.7.46	USDHC_SetStandardTuningCounter	1193
42.7.47	USDHC_EnableStandardTuning	1193
42.7.48	USDHC_GetExecuteStdTuningStatus	1194
42.7.49	USDHC_CheckStdTuningResult	1195
42.7.50	USDHC_CheckTuningError	1195
42.7.51	USDHC_EnableDDRMode	1195
42.7.52	USDHC_EnableHS400Mode	1195
42.7.53	USDHC_ResetStrobeDLL	1196
42.7.54	USDHC_EnableStrobeDLL	1196
42.7.55	USDHC_ConfigStrobeDLL	1196
42.7.56	USDHC_SetStrobeDllOverride	1196
42.7.57	USDHC_GetStrobeDLLStatus	1197
42.7.58	USDHC_SetDataConfig	1197

Section No.	Title	Page No.
42.7.59	USDHC_TransferCreateHandle	1197
42.7.60	USDHC_TransferNonBlocking	1198
42.7.61	USDHC_TransferBlocking	1198
42.7.62	USDHC_TransferHandleIRQ	1199
 Chapter 43 Debug Console		
43.1	Overview	1200
43.2	Function groups	1200
43.2.1	Initialization	1200
43.2.2	Advanced Feature	1201
43.2.3	SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART	1205
43.3	Typical use case	1206
43.4	Macro Definition Documentation	1208
43.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	1208
43.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	1208
43.4.3	DEBUGCONSOLE_DISABLE	1208
43.4.4	SDK_DEBUGCONSOLE	1208
43.4.5	PRINTF	1208
43.5	Function Documentation	1208
43.5.1	DbgConsole_Init	1208
43.5.2	DbgConsole_Deinit	1209
43.5.3	DbgConsole_EnterLowpower	1209
43.5.4	DbgConsole_ExitLowpower	1210
43.5.5	DbgConsole_Printf	1210
43.5.6	DbgConsole_Vprintf	1210
43.5.7	DbgConsole_Putchar	1210
43.5.8	DbgConsole_Scanf	1211
43.5.9	DbgConsole_Getchar	1211
43.5.10	DbgConsole_BlockingPrintf	1212
43.5.11	DbgConsole_BlockingVprintf	1212
43.5.12	DbgConsole_Flush	1212
43.5.13	DbgConsole_TryGetchar	1213
43.6	debug console configuration	1215
43.6.1	Overview	1215
43.6.2	Macro Definition Documentation	1216
43.7	Semihosting	1218
43.7.1	Guide Semihosting for IAR	1218
43.7.2	Guide Semihosting for Keil µVision	1218
43.7.3	Guide Semihosting for MCUXpresso IDE	1219

Section No.	Title	Page No.
43.7.4	Guide Semihosting for ARMGCC	1219
43.8	SWO	1222
43.8.1	Guide SWO for SDK	1222
43.8.2	Guide SWO for Keil µVision	1223
43.8.3	Guide SWO for MCUXpresso IDE	1224
43.8.4	Guide SWO for ARMGCC	1224

Chapter 44 Notification Framework

44.1	Overview	1225
44.2	Notifier Overview	1225
44.3	Data Structure Documentation	1227
44.3.1	struct notifier_notification_block_t	1227
44.3.2	struct notifier_callback_config_t	1228
44.3.3	struct notifier_handle_t	1228
44.4	Typedef Documentation	1229
44.4.1	notifier_user_config_t	1229
44.4.2	notifier_user_function_t	1229
44.4.3	notifier_callback_t	1230
44.5	Enumeration Type Documentation	1230
44.5.1	_notifier_status	1230
44.5.2	notifier_policy_t	1231
44.5.3	notifier_notification_type_t	1231
44.5.4	notifier_callback_type_t	1231
44.6	Function Documentation	1232
44.6.1	NOTIFIER_CreateHandle	1232
44.6.2	NOTIFIER_SwitchConfig	1233
44.6.3	NOTIFIER_GetErrorCallbackIndex	1234

Chapter 45 Shell

45.1	Overview	1235
45.2	Function groups	1235
45.2.1	Initialization	1235
45.2.2	Advanced Feature	1235
45.2.3	Shell Operation	1235
45.3	Data Structure Documentation	1237
45.3.1	struct shell_command_t	1237

Section No.	Title	Page No.
45.4 Macro Definition Documentation		1238
45.4.1 SHELL_NON_BLOCKING_MODE		1238
45.4.2 SHELL_AUTO_COMPLETE		1238
45.4.3 SHELL_BUFFER_SIZE		1238
45.4.4 SHELL_MAX_ARGS		1238
45.4.5 SHELL_HISTORY_COUNT		1238
45.4.6 SHELL_HANDLE_SIZE		1238
45.4.7 SHELL_USE_COMMON_TASK		1239
45.4.8 SHELL_TASK_PRIORITY		1239
45.4.9 SHELL_TASK_STACK_SIZE		1239
45.4.10 SHELL_HANDLE_DEFINE		1239
45.4.11 SHELL_COMMAND_DEFINE		1239
45.4.12 SHELL_COMMAND		1240
45.5 Typedef Documentation		1240
45.5.1 cmd_function_t		1240
45.6 Enumeration Type Documentation		1240
45.6.1 shell_status_t		1240
45.7 Function Documentation		1240
45.7.1 SHELL_Init		1240
45.7.2 SHELL_RegisterCommand		1241
45.7.3 SHELL_UnregisterCommand		1242
45.7.4 SHELL_Write		1242
45.7.5 SHELL_Printf		1243
45.7.6 SHELL_WriteSynchronization		1243
45.7.7 SHELL_PrintfSynchronization		1243
45.7.8 SHELL_ChangePrompt		1244
45.7.9 SHELL_PrintPrompt		1244
45.7.10 SHELL_Task		1244
45.7.11 SHELL_checkRunningInIsr		1245

Chapter 46 Cards: Secure Digital Card/Embedded MultiMedia Card/SDIO Card

46.1 Overview	1246
46.2 SDIO Card Driver	1247
46.2.1 Overview	1247
46.2.2 SDIO CARD Operation	1247
46.2.3 Data Structure Documentation	1249
46.2.4 Macro Definition Documentation	1251
46.2.5 Enumeration Type Documentation	1251
46.2.6 Function Documentation	1251

Section No.	Title	Page No.
46.3 SD Card Driver		1268
46.3.1 Overview		1268
46.3.2 SD CARD Operation		1268
46.3.3 Data Structure Documentation		1271
46.3.4 Macro Definition Documentation		1272
46.3.5 Enumeration Type Documentation		1272
46.3.6 Function Documentation		1272
46.4 MMC Card Driver		1282
46.4.1 Overview		1282
46.4.2 MMC CARD Operation		1282
46.4.3 Data Structure Documentation		1284
46.4.4 Macro Definition Documentation		1286
46.4.5 Enumeration Type Documentation		1286
46.4.6 Function Documentation		1287
46.5 SDMMC HOST Driver		1300
46.5.1 Overview		1300
46.6 SDMMC OSA		1301
46.6.1 Overview		1301
46.6.2 Data Structure Documentation		1302
46.6.3 Function Documentation		1302
46.6.4 USDHC HOST adapter Driver		1306
46.7 SDMMC Common		1319
46.7.1 Overview		1319
46.7.2 Data Structure Documentation		1338
46.7.3 Macro Definition Documentation		1351
46.7.4 Enumeration Type Documentation		1351
46.7.5 Function Documentation		1369
 Chapter 47 CODEC Driver		
47.1 Overview		1373
47.2 CODEC Common Driver		1374
47.2.1 Overview		1374
47.2.2 Data Structure Documentation		1379
47.2.3 Macro Definition Documentation		1380
47.2.4 Enumeration Type Documentation		1380
47.2.5 Function Documentation		1385
47.3 CODEC I2C Driver		1389
47.3.1 Overview		1389
47.3.2 Data Structure Documentation		1390

Section No.	Title	Page No.
47.3.3	Enumeration Type Documentation	1390
47.3.4	Function Documentation	1390
47.3.5	WM8962 Adapter	1393

Chapter 48 Serial Manager

48.1	Overview	1401
48.2	Data Structure Documentation	1404
48.2.1	struct serial_manager_config_t	1404
48.2.2	struct serial_manager_callback_message_t	1405
48.3	Macro Definition Documentation	1405
48.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE	1405
48.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE	1405
48.3.3	SERIAL_MANAGER_USE_COMMON_TASK	1405
48.3.4	SERIAL_MANAGER_HANDLE_SIZE	1405
48.3.5	SERIAL_MANAGER_HANDLE_DEFINE	1405
48.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	1406
48.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE	1406
48.3.8	SERIAL_MANAGER_TASK_PRIORITY	1407
48.3.9	SERIAL_MANAGER_TASK_STACK_SIZE	1407
48.4	Enumeration Type Documentation	1407
48.4.1	serial_port_type_t	1407
48.4.2	serial_manager_type_t	1407
48.4.3	serial_manager_status_t	1407
48.5	Function Documentation	1408
48.5.1	SerialManager_Init	1408
48.5.2	SerialManager_Deinit	1409
48.5.3	SerialManager_OpenWriteHandle	1409
48.5.4	SerialManager_CloseWriteHandle	1411
48.5.5	SerialManager_OpenReadHandle	1412
48.5.6	SerialManager_CloseReadHandle	1413
48.5.7	SerialManager_WriteBlocking	1413
48.5.8	SerialManager_ReadBlocking	1414
48.5.9	SerialManager_WriteNonBlocking	1415
48.5.10	SerialManager_ReadNonBlocking	1415
48.5.11	SerialManager_TryRead	1416
48.5.12	SerialManager_CancelWriting	1417
48.5.13	SerialManager_CancelReading	1417
48.5.14	SerialManager_InstallTxCallback	1418
48.5.15	SerialManager_InstallRxCallback	1418
48.5.16	SerialManager_needPollingIsr	1420

Section No.	Title	Page No.
48.5.17	SerialManager_EnterLowpower	1420
48.5.18	SerialManager_ExitLowpower	1420
48.5.19	SerialManager_SetLowpowerCriticalCb	1421
48.6	Serial Port Uart	1422
48.6.1	Overview	1422
48.6.2	Enumeration Type Documentation	1422
48.7	Serial Port USB	1423
48.7.1	Overview	1423
48.7.2	Data Structure Documentation	1424
48.7.3	Enumeration Type Documentation	1424
48.7.4	USB Device Configuration	1425
48.8	Serial Port SWO	1426
48.8.1	Overview	1426
48.8.2	Data Structure Documentation	1426
48.8.3	Enumeration Type Documentation	1426
Chapter 49 Gpc		
49.1	Overview	1427
49.2	Macro Definition Documentation	1429
49.2.1	FSL_GPC_RIVER_VERSION	1429
49.3	Enumeration Type Documentation	1429
49.3.1	anonymous enum	1429
49.3.2	anonymous enum	1430
49.3.3	gpc_cpu_slice_t	1430
49.3.4	gpc_cm_tran_step_t	1430
49.3.5	gpc_cpu_mode_t	1431
49.3.6	gpc_ss_tran_step_t	1431
49.4	Function Documentation	1431
49.4.1	GPC_CM_SetNextCpuMode	1431
49.4.2	GPC_CM_GetCurrentCpuMode	1432
49.4.3	GPC_CM_GetPreviousCpuMode	1432
49.4.4	GPC_CM_EnableIrqWakeups	1432
49.4.5	GPC_CM_EnableNonIrqWakeups	1433
49.4.6	GPC_CM_GetIrqWakeupsStatus	1433
49.4.7	GPC_CM_GetNonIrqWakeupsStatus	1433
49.4.8	GPC_CM_EnableCpuModeTransitionStep	1434
49.4.9	GPC_CM_RequestSystemSleepMode	1434
49.4.10	GPC_CM_ClearSystemSleepModeRequest	1434
49.4.11	GPC_CM_GetSystemSleepModeStatus	1434

Section No.	Title	Page No.
49.4.12	GPC_SS_EnableSystemSleepTransitionStep	1435
49.4.13	GPC_SS_SoftwareTriggerPMICStandby	1435
49.4.14	GPC_SS_SystemSleepTriggerPMICStandby	1435

Chapter 50 Asrc_edma_driver

50.1	Overview	1437
50.2	Data Structure Documentation	1438
50.2.1	struct asrc_p2p_edma_config_t	1438
50.2.2	struct asrc_in_edma_handle_t	1438
50.2.3	struct asrc_out_edma_handle_t	1439
50.2.4	struct _asrc_edma_handle	1440
50.3	Macro Definition Documentation	1440
50.3.1	ASRC_XFER_IN_QUEUE_SIZE	1440
50.4	Function Documentation	1440
50.4.1	ASRC_TransferInCreateHandleEDMA	1440
50.4.2	ASRC_TransferOutCreateHandleEDMA	1441
50.4.3	ASRC_TransferSetChannelPairConfigEDMA	1441
50.4.4	ASRC_GetOutSamplesSizeEDMA	1442
50.4.5	ASRC_TransferEDMA	1442
50.4.6	ASRC_TransferInAbortEDMA	1443
50.4.7	ASRC_TransferOutAbortEDMA	1443
50.4.8	ASRC_TransferInTerminalEDMA	1444
50.4.9	ASRC_TransferOutTerminalEDMA	1445

Chapter 51 I3c_master_edma_driver

51.1	Overview	1446
51.2	Data Structure Documentation	1446
51.2.1	struct i3c_master_edma_callback_t	1446
51.2.2	struct _i3c_master_edma_handle	1447
51.3	Function Documentation	1448
51.3.1	I3C_MasterTransferCreateHandleEDMA	1448
51.3.2	I3C_MasterTransferEDMA	1449
51.3.3	I3C_MasterTransferGetCountEDMA	1449
51.3.4	I3C_MasterTransferAbortEDMA	1450
51.3.5	I3C_MasterTransferEDMAHandleIRQ	1450

Section No.	Title	Page No.
Chapter 52 I3c_slave_edma_driver		
52.1 Overview	1451	
52.2 Data Structure Documentation	1451	
52.2.1 struct i3c_slave_edma_transfer_t	1451	
52.2.2 struct _i3c_slave_edma_handle	1452	
52.3 Typedef Documentation	1453	
52.3.1 i3c_slave_edma_callback_t	1453	
52.4 Function Documentation	1453	
52.4.1 I3C_SlaveTransferCreateHandleEDMA	1453	
52.4.2 I3C_SlaveTransferEDMA	1454	
52.4.3 I3C_SlaveTransferAbortEDMA	1455	
52.4.4 I3C_SlaveTransferEDMAHandleIRQ	1455	
Chapter 53 Xbar		
53.1 Overview	1456	
53.2 Data Structure Documentation	1457	
53.2.1 struct xbar_info_t	1457	
53.2.2 struct xbar_control_config_t	1457	
53.3 Enumeration Type Documentation	1457	
53.3.1 xbar_active_edge_t	1457	
53.3.2 xbar_request_t	1457	
53.4 Function Documentation	1457	
53.4.1 XBAR_Init	1457	
53.4.2 XBAR_Deinit	1458	
53.4.3 XBAR_SetSignalsConnection	1458	
53.4.4 XBAR_ClearOutputStatusFlag	1458	
53.4.5 XBAR_GetOutputStatusFlag	1459	
53.4.6 XBAR_SetOutputSignalConfig	1459	
53.5 Wm8962	1461	
53.5.1 Overview	1461	
53.5.2 Data Structure Documentation	1464	
53.5.3 Macro Definition Documentation	1466	
53.5.4 Enumeration Type Documentation	1466	
53.5.5 Function Documentation	1469	

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

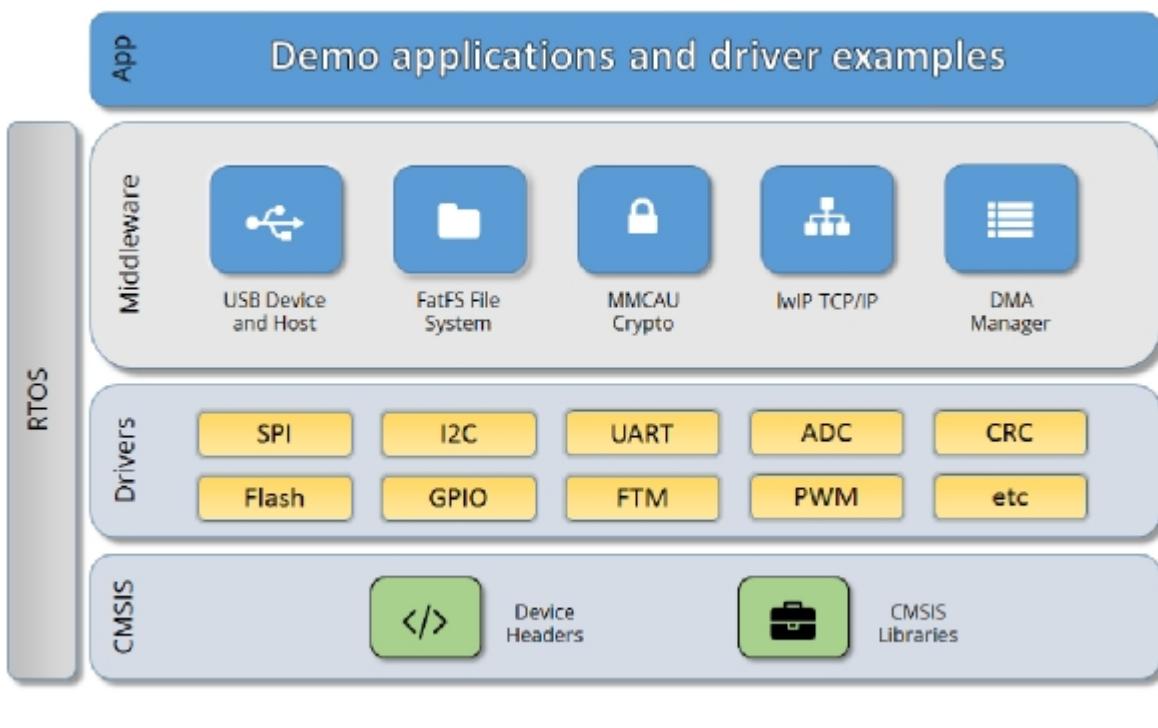
Architectural Overview

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

Files

- file [fsl_clock.h](#)

Data Structures

- struct [clock_arm_pll_config_t](#)
PLL configuration for ARM. [More...](#)
- struct [clock_usb_pll_config_t](#)
PLL configuration for USB. [More...](#)
- struct [clock_pll_ss_config_t](#)
Spread spectrum configuration for PLL. [More...](#)
- struct [clock_sys_pll2_config_t](#)
PLL configuration for Sys PLL2. [More...](#)
- struct [clock_sys_pll1_config_t](#)
PLL configuration for Sys PLL1. [More...](#)
- struct [clock_audio_pll_config_t](#)
PLL configuration for AUDIO. [More...](#)
- struct [clock_root_config_t](#)
Clock root configuration. [More...](#)

Macros

- #define [FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL](#) 0
Configure whether driver controls clock.
- #define [CCSR_OFFSET](#) 0x0C
CCM registers offset.
- #define [ARM_PLL_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 [PLL_SYS1_1G_FREQ](#) (1000000000UL)
SYS_PLL_FREQ frequency in Hz.
- #define [LPADC_CLOCKS](#)

- `#define AOI_CLOCKS`
Clock gate name array for LPADC.
- `#define ASRC_CLOCKS`
Clock ip name array for AOI.
- `#define CMP_CLOCKS`
Clock ip name array for ASRC.
- `#define DAC_CLOCKS`
Clock ip name array for CMP.
- `#define DCDC_CLOCKS`
Clock ip name array for DAC.
- `#define ECAT_CLOCKS`
Clock ip name array for DCDC.
- `#define EDMA_CLOCKS`
Clock ip name array for ECAT.
- `#define ENC_CLOCKS`
Clock gate name array for EDMA.
- `#define EWM_CLOCKS`
Clock ip name array for ENC.
- `#define FLEXCAN_CLOCKS`
Clock ip name array for EWM.
- `#define FLEXIO_CLOCKS`
Clock ip name array for FLEXCAN.
- `#define FLEXSPI_CLOCKS`
Clock ip name array for FLEXIO.
- `#define FLEXSPI_SLV_CLOCKS`
Clock gate name array for FLEXSPI.
- `#define GPC_CLOCKS`
Clock gate name array for FLEXSPI_SLV.
- `#define GPIO_CLOCKS`
Clock gate name array for GPC.
- `#define GPT_CLOCKS`
Clock ip name array for GPIO.
- `#define I3C_CLOCKS`
Clock ip name array for GPT.
- `#define IEE_CLOCKS`
Clock ip name array for I3C.
- `#define KPP_CLOCKS`
Clock ip name array for IEE.
- `#define LPI2C_CLOCKS`
Clock ip name array for KPP.
- `#define LPIT_CLOCKS`
Clock ip name array for LPI2C.
- `#define LPSPI_CLOCKS`
Clock ip name array for LPIT.
- `#define LPTMR_CLOCKS`
Clock ip name array for LPSPI.
- `#define LPUART_CLOCKS`
Clock ip name array for LPTMR.
- `#define PDM_CLOCKS`
Clock ip name array for LPUART.
- `#define MIC_CLOCKS`
Clock ip name array for MIC.

- #define **MU_CLOCKS**
Clock gate name array for MU.
- #define **NETC_CLOCKS**
Clock ip name array for NETC.
- #define **OCOTP_CLOCKS**
Clock ip name array for OCOTP.
- #define **PWM_CLOCKS**
Clock ip name array for PWM.
- #define **SAI_CLOCKS**
Clock ip name array for SAI.
- #define **SEMA42_CLOCKS**
Clock gate name array for Sema.
- #define **SEMC_CLOCKS**
Clock ip name array for SEMC.
- #define **SERDES_CLOCKS**
Clock ip name array for SERDES.
- #define **SINC_CLOCKS**
Clock ip name array for SINC.
- #define **SPDIF_CLOCKS**
Clock ip name array for SPDIF.
- #define **SRC_CLOCKS**
Clock gate name array for SRC.
- #define **TMR_CLOCKS**
Clock ip name array for QTIMER.
- #define **TPM_CLOCKS**
Clock ip name array for TPM.
- #define **USB_CLOCKS**
Clock ip name array for USB.
- #define **USDHC_CLOCKS**
Clock ip name array for USDHC.
- #define **WDOG_CLOCKS**
Clock gate name array for WDOG.
- #define **XBAR_CLOCKS**
Clock ip name array for XBAR.
- #define **CLOCK_GetCoreSysClkFreq** CLOCK_GetCpuClkFreq
For compatible with other platforms without CCM.

Enumerations

- enum `clock_lpcg_t` {
 `kCLOCK_M7` = 0,
 `kCLOCK_M33` = 1,
 `kCLOCK_Edgelock` = 2,
 `kCLOCK_Sim_Aon` = 3,
 `kCLOCK_Sim_Wakeup` = 4,
 `kCLOCK_Sim_Mega` = 5,
 `kCLOCK_Sim_R` = 6,
 `kCLOCK_Anadig` = 7,
 `kCLOCK_Dcdc` = 8,
 `kCLOCK_Src` = 9,
 `kCLOCK_Ccm` = 10,
 `kCLOCK_Gpc` = 11,
 `kCLOCK_Adcl` = 12,
 `kCLOCK_Adc2` = 13,
 `kCLOCK_Dac` = 14,
 `kCLOCK_Acmp1` = 15,
 `kCLOCK_Acmp2` = 16,
 `kCLOCK_Acmp3` = 17,
 `kCLOCK_Acmp4` = 18,
 `kCLOCK_Wdog1` = 19,
 `kCLOCK_Wdog2` = 20,
 `kCLOCK_Wdog3` = 21,
 `kCLOCK_Wdog4` = 22,
 `kCLOCK_Wdog5` = 23,
 `kCLOCK_Ewm0` = 24,
 `kCLOCK_Sema1` = 25,
 `kCLOCK_Sema2` = 26,
 `kCLOCK_Mu_A` = 27,
 `kCLOCK_Mu_B` = 28,
 `kCLOCK_Edma1` = 29,
 `kCLOCK_Edma2` = 30,
 `kCLOCK_Romcp` = 31,
 `kCLOCK_Oram1` = 32,
 `kCLOCK_Oram2` = 33,
 `kCLOCK_Flexspi1` = 34,
 `kCLOCK_Flexspi2` = 35,
 `kCLOCK_Flexspi_Slv` = 36,
 `kCLOCK_Trdc` = 37,
 `kCLOCK_Ocotp` = 38,
 `kCLOCK_Semc` = 39,
 `kCLOCK_Iee` = 40,
 `kCLOCK_Cstrace` = 41,
 `kCLOCK_Csswo` = 42,
 `kCLOCK_Iomuxc1` = 43,
 `kCLOCK_Semc_Iomuxc2` = 44,
 `kCLOCK_Gpio1` = 45,
 `kCLOCK_Gpio2` = 46,

- ```
kCLOCK_IpInvalid }
```
- Clock LPCG index.*
- enum `clock_name_t` {  
 kCLOCK\_OscRc24M = 0,  
 kCLOCK\_OscRc400M = 1,  
 kCLOCK\_Osc24M = 2,  
 kCLOCK\_Osc24MOut = 3,  
 kCLOCK\_ArmPll = 4,  
 kCLOCK\_ArmPllOut = 5,  
 kCLOCK\_SysPll2 = 6,  
 kCLOCK\_SysPll2Out = 7,  
 kCLOCK\_SysPll2Pfd0 = 8,  
 kCLOCK\_SysPll2Pfd1 = 9,  
 kCLOCK\_SysPll2Pfd2 = 10,  
 kCLOCK\_SysPll2Pfd3 = 11,  
 kCLOCK\_SysPll3 = 12,  
 kCLOCK\_SysPll3Out = 13,  
 kCLOCK\_SysPll3Div2 = 14,  
 kCLOCK\_SysPll3Pfd0 = 15,  
 kCLOCK\_SysPll3Pfd1 = 16,  
 kCLOCK\_SysPll3Pfd2 = 17,  
 kCLOCK\_SysPll3Pfd3 = 18,  
 kCLOCK\_SysPll1 = 19,  
 kCLOCK\_SysPll1Out = 20,  
 kCLOCK\_SysPll1Div2 = 21,  
 kCLOCK\_SysPll1Div5 = 22,  
 kCLOCK\_AudioPll = 23,  
 kCLOCK\_AudioPllOut = 24,  
 kCLOCK\_CpuClk,  
 kCLOCK\_CoreSysClk }
- Clock name.*
- enum `clock_root_t` {

kCLOCK\_Root\_M7 = 0,  
kCLOCK\_Root\_M33 = 1,  
kCLOCK\_Root\_Edgelock = 2,  
kCLOCK\_Root\_Bus\_Aon = 3,  
kCLOCK\_Root\_Bus\_Wakeup = 4,  
kCLOCK\_Root\_Wakeup\_Axi = 5,  
kCLOCK\_Root\_Swo\_Trace = 6,  
kCLOCK\_Root\_M33\_Systick = 7,  
kCLOCK\_Root\_M7\_Systick = 8,  
kCLOCK\_Root\_Flexio1 = 9,  
kCLOCK\_Root\_Flexio2 = 10,  
kCLOCK\_Root\_Lpit3 = 11,  
kCLOCK\_Root\_Lptimer1 = 12,  
kCLOCK\_Root\_Lptimer2 = 13,  
kCLOCK\_Root\_Lptimer3 = 14,  
kCLOCK\_Root\_Tpm2 = 15,  
kCLOCK\_Root\_Tpm4 = 16,  
kCLOCK\_Root\_Tpm5 = 17,  
kCLOCK\_Root\_Tpm6 = 18,  
kCLOCK\_Root\_Gpt1 = 19,  
kCLOCK\_Root\_Gpt2 = 20,  
kCLOCK\_Root\_Flexspi1 = 21,  
kCLOCK\_Root\_Flexspi2 = 22,  
kCLOCK\_Root\_Flexspi\_Slv = 23,  
kCLOCK\_Root\_Can1 = 24,  
kCLOCK\_Root\_Can2 = 25,  
kCLOCK\_Root\_Can3 = 26,  
kCLOCK\_Root\_Lpuart0102 = 27,  
kCLOCK\_Root\_Lpuart0304 = 28,  
kCLOCK\_Root\_Lpuart0506 = 29,  
kCLOCK\_Root\_Lpuart0708 = 30,  
kCLOCK\_Root\_Lpuart0910 = 31,  
kCLOCK\_Root\_Lpuart1112 = 32,  
kCLOCK\_Root\_Lpi2c0102 = 33,  
kCLOCK\_Root\_Lpi2c0304 = 34,  
kCLOCK\_Root\_Lpi2c0506 = 35,  
kCLOCK\_Root\_Lpspi0102 = 36,  
kCLOCK\_Root\_Lpspi0304 = 37,  
kCLOCK\_Root\_Lpspi0506 = 38,  
kCLOCK\_Root\_I3c1 = 39,  
kCLOCK\_Root\_I3c2 = 40,  
kCLOCK\_Root\_Usdhc1 = 41,  
kCLOCK\_Root\_Usdhc2 = 42,  
kCLOCK\_Root\_Semc = 43,  
kCLOCK\_Root\_Adcl = 44,  
kCLOCK\_Root\_Adc2 = 45,  
kCLOCK\_Root\_Acmp\_M16 = 46,  
kCLOCK\_Root\_Ecat = 47,  
kCLOCK\_Root\_Enet = 48,

```
kCLOCK_Root_Cko2 = 73 }
 Root clock index.
• enum clock_root_mux_source_t {
```

kCLOCK\_M7\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_M7\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_M7\_ClockRoot\_MuxArmPllOut = 2U,  
 kCLOCK\_M7\_ClockRoot\_MuxSysPll3Out = 3U,  
 kCLOCK\_M33\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_M33\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_M33\_ClockRoot\_MuxSysPll3Out = 2U,  
 kCLOCK\_M33\_ClockRoot\_MuxArmPllOut = 3U,  
 kCLOCK\_EDGELOCK\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_EDGELOCK\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_EDGELOCK\_ClockRoot\_MuxSysPll1Out = 2U,  
 kCLOCK\_EDGELOCK\_ClockRoot\_MuxSysPll2Pfd1 = 3U,  
 kCLOCK\_BUS\_AON\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_BUS\_AON\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_BUS\_AON\_ClockRoot\_MuxSysPll2Out = 2U,  
 kCLOCK\_BUS\_AON\_ClockRoot\_MuxSysPll3Pfd2 = 3U,  
 kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxSysPll2Out = 2U,  
 kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxSysPll3Pfd1 = 3U,  
 kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxSysPll3Out = 2U,  
 kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxSysPll2Pfd1 = 3U,  
 kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxSysPll3Div2 = 2U,  
 kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxSysPll1Div5 = 3U,  
 kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxOsc24MOut = 2U,  
 kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxSysPll3Div2 = 3U,  
 kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxOsc24MOut = 2U,  
 kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxSysPll3Div2 = 3U,  
 kCLOCK\_FLEXIO1\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_FLEXIO1\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_FLEXIO1\_ClockRoot\_MuxSysPll3Div2 = 2U,  
 kCLOCK\_FLEXIO1\_ClockRoot\_MuxSysPll1Div5 = 3U,  
 kCLOCK\_FLEXIO2\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_FLEXIO2\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_FLEXIO2\_ClockRoot\_MuxSysPll3Div2 = 2U,  
 kCLOCK\_FLEXIO2\_ClockRoot\_MuxSysPll1Div5 = 3U,  
 kCLOCK\_LPIT3\_ClockRoot\_MuxOscRc24M = 0U,  
 kCLOCK\_LPIT3\_ClockRoot\_MuxOscRc400M = 1U,  
 kCLOCK\_LPIT3\_ClockRoot\_MuxSysPll2Pfd3 = 3U,  
 kCLOCK\_LPTIMER1\_ClockRoot\_MuxOscRc24M = 0U,

`kCLOCK_CKO2_ClockRoot_MuxArmPllOut = 3U }`

*The enumerator of clock roots' clock source mux value.*

- enum `clock_osc_t` {
   
`kCLOCK_RcOsc = 0U,`
  
`kCLOCK_XtalOsc = 1U }`

*OSC 24M sorce select.*
- enum `clock_gate_value_t` {
   
`kCLOCK_Off = (int)~CCM_LPCG_DIRECT_ON_MASK,`
  
`kCLOCK_On = CCM_LPCG_DIRECT_ON_MASK }`

*Clock gate value.*
- enum `clock_mode_t` {
   
`kCLOCK_ModeRun = 0U,`
  
`kCLOCK_ModeWait = 1U,`
  
`kCLOCK_ModeStop = 2U }`

*System clock mode.*
- enum `clock_usb_src_t` {
   
`kCLOCK_Usb480M = 0,`
  
`kCLOCK_UsbSrcUnused = (int)0xFFFFFFFFU }`

*USB clock source definition.*
- enum `clock_usb_phy_src_t` { `kCLOCK_Usbphy480M = 0` }
 

*Source of the USB HS PHY.*
- enum `_clock_pll_clk_src` {
   
`kCLOCK_PlIclkSrc24M = 0U,`
  
`kCLOCK_PlISrcClkPN = 1U }`

*PLL clock source, bypass cloco source also.*
- enum `clock_pll_post_div_t` {
   
`kCLOCK_PlIPostDiv2 = 0U,`
  
`kCLOCK_PlIPostDiv4 = 1U,`
  
`kCLOCK_PlIPostDiv8 = 2U,`
  
`kCLOCK_PlIPostDiv1 = 3U }`

*PLL post divider enumeration.*
- enum `clock_pll_t` {
   
`kCLOCK_PlIArm,`
  
`kCLOCK_PlISys1,`
  
`kCLOCK_PlISys2,`
  
`kCLOCK_PlISys3,`
  
`kCLOCK_PlIAudio,`
  
`kCLOCK_PlIInvalid = -1 }`

*PLL name.*
- enum `clock_pfd_t` {
   
`kCLOCK_Pfd0 = 0U,`
  
`kCLOCK_Pfd1 = 1U,`
  
`kCLOCK_Pfd2 = 2U,`
  
`kCLOCK_Pfd3 = 3U }`

*PLL PFD name.*
- enum `clock_control_mode_t` {
   
`kCLOCK_SoftwareMode = 0U,`

- `kCLOCK_GpcMode }`  
*The enumeration of control mode.*
- enum `clock_24MOsc_mode_t` {
   
`kCLOCK_24MOscHighGainMode` = 0U,  
`kCLOCK_24MOscBypassMode` = 1U,  
`kCLOCK_24MOscLowPowerMode` = 2U }
   
*The enumeration of 24MHz crystal oscillator mode.*
- enum `clock_1MHzOut_behavior_t` {
   
`kCLOCK_1MHzOutDisable` = 0U,  
`kCLOCK_1MHzOutEnableLocked1Mhz` = 1U,  
`kCLOCK_1MHzOutEnableFreeRunning1Mhz` = 2U }
   
*The enumeration of 1MHz output clock behavior, including disabling 1MHz output, enabling locked 1MHz clock output, and enabling free-running 1MHz clock output.*
- enum `clock_level_t` {
   
`kCLOCK_Level0` = 0x0UL,  
`kCLOCK_Level1` = 0x1UL,  
`kCLOCK_Level2` = 0x2UL,  
`kCLOCK_Level3` = 0x3UL,  
`kCLOCK_Level4` = 0x4UL }
   
*The clock dependence level.*

## Functions

- static void `CLOCK_SetRootClockMux` (`clock_root_t` root, `uint8_t` src)
   
*Set CCM Root Clock MUX node to certain value.*
- static `uint32_t` `CLOCK_GetRootClockMux` (`clock_root_t` root)
   
*Get CCM Root Clock MUX value.*
- static `clock_name_t` `CLOCK_GetRootClockSource` (`clock_root_t` root, `uint32_t` src)
   
*Get CCM Root Clock Source.*
- static void `CLOCK_SetRootClockDiv` (`clock_root_t` root, `uint32_t` div)
   
*Set CCM Root Clock DIV certain value.*
- static `uint32_t` `CLOCK_GetRootClockDiv` (`clock_root_t` root)
   
*Get CCM DIV node value.*
- static void `CLOCK_PowerOffRootClock` (`clock_root_t` root)
   
*Power Off Root Clock.*
- static void `CLOCK_PowerOnRootClock` (`clock_root_t` root)
   
*Power On Root Clock.*
- static void `CLOCK_SetRootClock` (`clock_root_t` root, const `clock_root_config_t` \*config)
   
*Configure Root Clock.*
- static void `CLOCK_ControlGate` (`clock_ip_name_t` name, `clock_gate_value_t` value)
   
*Control the clock gate for specific IP.*
- static void `CLOCK_EnableClock` (`clock_ip_name_t` name)
   
*Enable the clock for specific IP.*
- static void `CLOCK_DisableClock` (`clock_ip_name_t` name)
   
*Disable the clock for specific IP.*
- `uint32_t` `CLOCK_GetFreq` (`clock_name_t` name)
   
*Gets the clock frequency for a specific clock name.*
- static `uint32_t` `CLOCK_GetRootClockFreq` (`clock_root_t` root)
   
*Gets the clock frequency for a specific root clock name.*
- `uint32_t` `CLOCK_GetM7Freq` (void)

- `uint32_t CLOCK_GetM33Freq (void)`  
*Get the CCM CPU/core/system frequency.*
- `static bool CLOCK_IsPllBypassed (clock_pll_t pll)`  
*Check if PLL is bypassed.*
- `static bool CLOCK_IsPllEnabled (clock_pll_t pll)`  
*Check if PLL is enabled.*
- `void CLOCK_InitArmPll (const clock_arm_pll_config_t *config)`  
*Initialize the ARM PLL.*
- `status_t CLOCK_CalcArmPllFreq (clock_arm_pll_config_t *config, uint32_t freqInMhz)`  
*Calculate corresponding config values per given frequency.*
- `status_t CLOCK_InitArmPllWithFreq (uint32_t freqInMhz)`  
*Initializes the Arm PLL with Specific Frequency (in Mhz).*
- `void CLOCK_DeinitArmPll (void)`  
*De-initialize the ARM PLL.*
- `void CLOCK_CalcPllSpreadSpectrum (uint32_t factor, uint32_t range, uint32_t mod, clock_pll_ss_config_t *ss)`  
*Calculate spread spectrum step and stop.*
- `void CLOCK_InitSysPll1 (const clock_sys_pll1_config_t *config)`  
*Initialize the System PLL1.*
- `void CLOCK_DeinitSysPll1 (void)`  
*De-initialize the System PLL1.*
- `void CLOCK_InitSysPll2 (const clock_sys_pll2_config_t *config)`  
*Initialize the System PLL2.*
- `void CLOCK_DeinitSysPll2 (void)`  
*De-initialize the System PLL2.*
- `bool CLOCK_IsSysPll2PfdEnabled (clock_pfd_t pfd)`  
*Check if Sys PLL2 PFD is enabled.*
- `void CLOCK_InitSysPll3 (void)`  
*Initialize the System PLL3.*
- `void CLOCK_DeinitSysPll3 (void)`  
*De-initialize the System PLL3.*
- `bool CLOCK_IsSysPll3PfdEnabled (clock_pfd_t pfd)`  
*Check if Sys PLL3 PFD is enabled.*

## Driver version

- `#define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))`  
*CLOCK driver version.*
- `#define SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY (240000000UL)`

## OSC operations

- `static uint32_t CLOCK_GetRtcFreq (void)`  
*Gets the RTC clock frequency.*
- `void CLOCK_SetClockSourceControlMode (clock_name_t name, clock_control_mode_t controlMode)`  
*Set the control mode of a specified clock.*
- `static void CLOCK_OSC_EnableOscRc24M (bool enable)`  
*Enable/disable 24MHz RC oscillator.*
- `void CLOCK_OSC_EnableOsc24M (void)`

*Enable OSC 24Mhz.*

- static void **CLOCK\_OSC\_GateOsc24M** (bool enableGate)
 

*Gate/ungate the 24MHz crystal oscillator output.*
- void **CLOCK\_OSC\_SetOsc24MWorkMode** (*clock\_24MOsc\_mode\_t* workMode)
 

*Set the work mode of 24MHz crystal oscillator, the available modes are high gian mode, low power mode, and bypass mode.*
- void **CLOCK\_OSC\_EnableOscRc400M** (void)
 

*Enable OSC RC 400Mhz.*
- static void **CLOCK\_OSC\_GateOscRc400M** (bool enableGate)
 

*Gate/ungate 400MHz RC oscillator.*
- void **CLOCK\_OSC\_TrimOscRc400M** (bool enable, bool bypass, uint16\_t trim)
 

*Trims OSC RC 400MHz.*
- void **CLOCK\_OSC\_SetOscRc400MRefClkDiv** (uint8\_t divValue)
 

*Set the divide value for ref\_clk to generate slow clock.*
- void **CLOCK\_OSC\_SetOscRc400MFastClkCount** (uint16\_t targetCount)
 

*Set the target count for the fast clock.*
- void **CLOCK\_OSC\_SetOscRc400MHysteresisValue** (uint8\_t negHysteresis, uint8\_t posHysteresis)
 

*Set the negative and positive hysteresis value for the tuned clock.*
- void **CLOCK\_OSC\_BypassOscRc400MTuneLogic** (bool enableBypass)
 

*Bypass/un-bypass the tune logic.*
- void **CLOCK\_OSC\_EnableOscRc400MTuneLogic** (bool enable)
 

*Start/Stop the tune logic.*
- void **CLOCK\_OSC\_FreezeOscRc400MTuneValue** (bool enableFreeze)
 

*Freeze/Unfreeze the tuning value.*
- void **CLOCK\_OSC\_SetOscRc400MTuneValue** (uint8\_t tuneValue)
 

*Set the 400MHz RC oscillator tune value when the tune logic is disabled.*
- void **CLOCK\_OSC\_Set1MHzOutputBehavior** (*clock\_1MHzOut\_behavior\_t* behavior)
 

*Set the behavior of the 1MHz output clock, such as disable the 1MHz clock output, enable the free-running 1MHz clock output, enable the locked 1MHz clock output.*
- void **CLOCK\_OSC\_SetLocked1MHzCount** (uint16\_t count)
 

*Set the count for the locked 1MHz clock out.*
- bool **CLOCK\_OSC\_CheckLocked1MHzErrorFlag** (void)
 

*Check the error flag for locked 1MHz clock out.*
- void **CLOCK\_OSC\_ClearLocked1MHzErrorFlag** (void)
 

*Clear the error flag for locked 1MHz clock out.*
- uint16\_t **CLOCK\_OSC\_GetCurrentOscRc400MFastClockCount** (void)
 

*Get current count for the fast clock during the tune process.*
- uint8\_t **CLOCK\_OSC\_GetCurrentOscRc400MTuneValue** (void)
 

*Get current tune value used by oscillator during tune process.*

## PLL/PFD operations

- void **CLOCK\_SetPllBypass** (*clock\_pll\_t* pll, bool bypass)
 

*PLL bypass setting.*
- status\_t **CLOCK\_CalcAudioPllFreq** (*clock\_audio\_pll\_config\_t* \*config, uint32\_t freqInMhz)
 

*Calculate corresponding config values per given frequency.*
- status\_t **CLOCK\_InitAudioPllWithFreq** (uint32\_t freqInMhz, bool ssEnable, uint32\_t ssRange, uint32\_t ssMod)
 

*Initializes the Audio PLL with Specific Frequency (in Mhz).*
- void **CLOCK\_InitAudioPll** (const *clock\_audio\_pll\_config\_t* \*config)
 

*Initializes the Audio PLL.*

- void **CLOCK\_DeinitAudioPll** (void)  
*De-initialize the Audio PLL.*
- uint32\_t **CLOCK\_GetPllFreq** (**clock\_pll\_t** pll)  
*Get current PLL output frequency.*
- void **CLOCK\_InitPfd** (**clock\_pll\_t** pll, **clock\_pfd\_t** pfd, uint8\_t frac)  
*Initialize PLL PFD.*
- void **CLOCK\_DeinitPfd** (**clock\_pll\_t** pll, **clock\_pfd\_t** pfd)  
*De-initialize selected PLL PFD.*
- uint32\_t **CLOCK\_GetPfdFreq** (**clock\_pll\_t** pll, **clock\_pfd\_t** pfd)  
*Get current PFD output frequency.*
- uint32\_t **CLOCK\_GetFreqFromObs** (uint8\_t obsIndex, uint32\_t obsSigIndex)
- bool **CLOCK\_EnableUsbhs0Clock** (**clock\_usb\_src\_t** src, uint32\_t freq)  
*Enable USB HS clock.*
- bool **CLOCK\_EnableUsbhs1Clock** (**clock\_usb\_src\_t** src, uint32\_t freq)  
*Enable USB HS clock.*
- bool **CLOCK\_EnableUsbhs0PhyPllClock** (**clock\_usb\_phy\_src\_t** src, uint32\_t freq)  
*Enable USB HS PHY PLL clock.*
- void **CLOCK\_DisableUsbhs0PhyPllClock** (void)  
*Disable USB HS PHY PLL clock.*
- bool **CLOCK\_EnableUsbhs1PhyPllClock** (**clock\_usb\_phy\_src\_t** src, uint32\_t freq)  
*Enable USB HS PHY PLL clock.*
- void **CLOCK\_DisableUsbhs1PhyPllClock** (void)  
*Disable USB HS PHY PLL clock.*
- static void **CLOCK\_OSCPLL\_LockWhiteList** (**clock\_name\_t** name)  
*Lock the value of Domain ID white list for this clock.*
- static void **CLOCK\_OSCPLL\_SetWhiteList** (**clock\_name\_t** name, uint8\_t domainId)  
*Set domain ID that can change this clock.*
- void **CLOCK\_OSCPLL\_ControlByCpuLowPowerMode** (**clock\_name\_t** name, uint32\_t domainMap, **clock\_level\_t** level)  
*Set this clock works in CPU Low Power Mode.*
- static void **CLOCK\_ROOT\_LockWhiteList** (**clock\_root\_t** name)  
*Lock the value of Domain ID white list for this clock.*
- static void **CLOCK\_ROOT\_SetWhiteList** (**clock\_root\_t** name, uint8\_t domainId)  
*Set domain ID that can change this clock.*
- static void **CLOCK\_LPCG\_LockWhiteList** (**clock\_lpcg\_t** name)  
*Lock the value of Domain ID white list for this clock.*
- static void **CLOCK\_LPCG\_SetWhiteList** (**clock\_lpcg\_t** name, uint8\_t domainId)  
*Set domain ID that can change this clock.*
- void **CLOCK\_LPCG\_ControlByCpuLowPowerMode** (**clock\_lpcg\_t** name, uint32\_t domainMap, **clock\_level\_t** level)  
*Set this clock works in CPU Low Power Mode.*

## 4.2 Data Structure Documentation

### 4.2.1 struct **clock\_arm\_pll\_config\_t**

The output clock frequency is:

$F_{out} = F_{in} * \text{loopDivider} / (2 * \text{postDivider})$ .

$F_{in}$  is always 24MHz.

## Data Fields

- `clock_pll_post_div_t` `postDivider`  
*Post divider.*
- `uint32_t` `loopDivider`  
*PLL loop divider.*

### Field Documentation

(1) `clock_pll_post_div_t` `clock_arm_pll_config_t::postDivider`

(2) `uint32_t` `clock_arm_pll_config_t::loopDivider`

Valid range: 104-208.

## 4.2.2 struct `clock_usb_pll_config_t`

### Data Fields

- `uint8_t` `loopDivider`  
*PLL loop divider.*
- `uint8_t` `src`  
*PLL clock source, reference \_clock\_pll\_clk\_src.*

### Field Documentation

(1) `uint8_t` `clock_usb_pll_config_t::loopDivider`

0 - Fout=Fref\*20; 1 - Fout=Fref\*22

## 4.2.3 struct `clock_pll_ss_config_t`

### Data Fields

- `uint16_t` `stop`  
*Spread spectrum stop value to get frequency change.*
- `uint16_t` `step`  
*Spread spectrum step value to get frequency change step.*

**Field Documentation**

- (1) `uint16_t clock_pll_ss_config_t::stop`
- (2) `uint16_t clock_pll_ss_config_t::step`

**4.2.4 struct clock\_sys\_pll2\_config\_t****Data Fields**

- `uint32_t mfd`  
*Denominator of spread spectrum.*
- `clock_pll_ss_config_t * ss`  
*Spread spectrum parameter, it can be NULL, if ssEnable is set to false.*
- `bool ssEnable`  
*Enable spread spectrum flag.*

**4.2.5 struct clock\_sys\_pll1\_config\_t****Data Fields**

- `bool pllDiv2En`  
*Enable Sys Pll1 divide-by-2 clock or not.*
- `bool pllDiv5En`  
*Enable Sys Pll1 divide-by-5 clock or not.*
- `clock_pll_ss_config_t * ss`  
*Spread spectrum parameter, it can be NULL, if ssEnable is set to false.*
- `bool ssEnable`  
*Enable spread spectrum flag.*

**Field Documentation**

- (1) `bool clock_sys_pll1_config_t::pllDiv2En`
- (2) `bool clock_sys_pll1_config_t::pllDiv5En`

**4.2.6 struct clock\_audio\_pll\_config\_t****Data Fields**

- `uint8_t loopDivider`  
*PLL loop divider.*
- `uint8_t postDivider`  
*Divider after the PLL, 0x0=divided by 1, 0x1=divided by 2, 0x2=divided by 4, 0x3=divided by 8, 0x4=divided by 16, 0x5=divided by 32.*
- `uint32_t numerator`  
*30 bit numerator of fractional loop divider.*

- `uint32_t denominator`  
*30 bit denominator of fractional loop divider*
- `clock_pll_ss_config_t * ss`  
*Spread spectrum parameter, it can be NULL, if ssEnable is set to false.*
- `bool ssEnable`  
*Enable spread spectrum flag.*

**Field Documentation**(1) `uint8_t clock_audio_pll_config_t::loopDivider`

Valid range for DIV\_SELECT divider value: 27~54.

(2) `uint8_t clock_audio_pll_config_t::postDivider`(3) `uint32_t clock_audio_pll_config_t::numerator`**4.2.7 struct clock\_root\_config\_t****Data Fields**

- `uint8_t mux`  
*See `clock_root_mux_source_t` for details.*
- `uint8_t div`  
*it's the actual divider*

**Field Documentation**(1) `uint8_t clock_root_config_t::mux`**4.3 Macro Definition Documentation****4.3.1 #define FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL 0**

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

**Note**

All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

**4.3.2 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))****4.3.3 #define LPADC\_CLOCKS****Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Adcl,
 kCLOCK_Adcl \
}
```

**4.3.4 #define AOI\_CLOCKS****Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Aoi1,
 kCLOCK_Aoi2, kCLOCK_Aoi3, kCLOCK_Aoi4 \
}
```

**4.3.5 #define ASRC\_CLOCKS****Value:**

```
{
 kCLOCK_Asrc \
}
```

**4.3.6 #define CMP\_CLOCKS****Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Acmpl,
 kCLOCK_Acmp2, kCLOCK_Acmp3, kCLOCK_Acmp4 \
}
```

#### 4.3.7 #define DAC\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Dac \
}
```

#### 4.3.8 #define DCDC\_CLOCKS

**Value:**

```
{ \
 kCLOCK_Dcdc \
}
```

#### 4.3.9 #define ECAT\_CLOCKS

**Value:**

```
{ \
 kCLOCK_ECAT \
}
```

#### 4.3.10 #define EDMA\_CLOCKS

**Value:**

```
{ \
 kCLOCK_IpInvalid, kCLOCK_Edma1,
 kCLOCK_Edma2 \
}
```

#### 4.3.11 #define ENC\_CLOCKS

**Value:**

```
{ \
 kCLOCK_IpInvalid, kCLOCK_Enc1,
 kCLOCK_Enc2, kCLOCK_Enc3, kCLOCK_Enc4 \
}
```

#### 4.3.12 #define EWM\_CLOCKS

**Value:**

```
{
 kCLOCK_Ewm0 \
}
```

#### 4.3.13 #define FLEXCAN\_CLOCKS

**Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Can1,
 kCLOCK_Can2, kCLOCK_Can3 \
}
```

#### 4.3.14 #define FLEXIO\_CLOCKS

**Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Flexio1,
 kCLOCK_Flexio2 \
}
```

#### 4.3.15 #define FLEXSPI\_CLOCKS

**Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Flexspi1,
 kCLOCK_Flexspi2 \
}
```

#### 4.3.16 #define FLEXSPI\_SLV\_CLOCKS

**Value:**

```
{
 kCLOCK_Flexspi_Slv \
}
```

#### 4.3.17 #define GPC\_CLOCKS

**Value:**

```
{
 kCLOCK_Gpc \
}
```

#### 4.3.18 #define GPIO\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_IpInvalid, kCLOCK_Gpio1,
 kCLOCK_Gpio2, kCLOCK_Gpio3, kCLOCK_Gpio4,
 kCLOCK_Gpio5, kCLOCK_Gpio6 \
}
```

#### 4.3.19 #define GPT\_CLOCKS

**Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Gpt1, \
 kCLOCK_Gpt2 \
}
```

#### 4.3.20 #define I3C\_CLOCKS

**Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_I3c1, \
 kCLOCK_I3c2 \
}
```

#### 4.3.21 #define IEE\_CLOCKS

**Value:**

```
{
 kCLOCK_Iee \
}
```

### 4.3.22 #define KPP\_CLOCKS

**Value:**

```
{
 \KCLOCK_Kpp \
}
```

### 4.3.23 #define LPI2C\_CLOCKS

**Value:**

```
{
 \
 \KCLOCK_IpInvalid, \KCLOCK_Lpi2c1,
 \KCLOCK_Lpi2c2, \KCLOCK_Lpi2c3, \KCLOCK_Lpi2c4,
 \KCLOCK_Lpi2c5, \KCLOCK_Lpi2c6 \
}
```

### 4.3.24 #define LPIT\_CLOCKS

**Value:**

```
{
 \KCLOCK_IpInvalid, \KCLOCK_Lpit1,
 \KCLOCK_Lpit2, \KCLOCK_Lpit3 \
}
```

### 4.3.25 #define LPSPI\_CLOCKS

**Value:**

```
{
 \
 \KCLOCK_IpInvalid, \KCLOCK_Lpspi1,
 \KCLOCK_Lpspi2, \KCLOCK_Lpspi3, \KCLOCK_Lpspi4,
 \KCLOCK_Lpspi5, \KCLOCK_Lpspi6 \
}
```

### 4.3.26 #define LPTMR\_CLOCKS

**Value:**

```
{
 \KCLOCK_IpInvalid, \KCLOCK_Lptmr1,
 \KCLOCK_Lptmr2, \KCLOCK_Lptmr3 \
}
```

### 4.3.27 #define LPUART\_CLOCKS

**Value:**

```
{\n \\ \n kCLOCK_IpInvalid, kCLOCK_Lpuart1,\n kCLOCK_Lpuart2, kCLOCK_Lpuart3, kCLOCK_Lpuart4,\n kCLOCK_Lpuart5, \\ \n kCLOCK_Lpuart6, kCLOCK_Lpuart7,\n kCLOCK_Lpuart8, kCLOCK_Lpuart9, kCLOCK_Lpuart10,\n kCLOCK_Lpuart11, \\ \n kCLOCK_Lpuart12\n \\ \n}
```

### 4.3.28 #define PDM\_CLOCKS

**Value:**

```
{\n \\ \n kCLOCK_Pdm \\ \n}
```

### 4.3.29 #define MU\_CLOCKS

**Value:**

```
{\n \\ \n kCLOCK_Mu_B, kCLOCK_Mu_B \\ \n}
```

### 4.3.30 #define NETC\_CLOCKS

**Value:**

```
{\n \\ \n kCLOCK_Netc \\ \n}
```

### 4.3.31 #define OCOTP\_CLOCKS

**Value:**

```
{\n \\ \n kCLOCK_Ocotp \\ \n}
```

### 4.3.32 #define PWM\_CLOCKS

**Value:**

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

### 4.3.33 #define SAI\_CLOCKS

**Value:**

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

### 4.3.34 #define SEMA42\_CLOCKS

**Value:**

```
{
 kCLOCK_IpInvalid, kCLOCK_Sema1,
 kCLOCK_Sema2 \
}
```

### 4.3.35 #define SEMC\_CLOCKS

**Value:**

```
{
 kCLOCK_Semc \
}
```

### 4.3.36 #define SERDES\_CLOCKS

**Value:**

```
{
 kCLOCK_Invalid, kCLOCK_Serdes1,
 kCLOCK_Serdes2, kCLOCK_Serdes3 \
}
```

### 4.3.37 #define SINC\_CLOCKS

**Value:**

```
{
 kCLOCK_Invalid, kCLOCK_Sinc1,
 kCLOCK_Sinc2, kCLOCK_Sinc3 \
}
```

### 4.3.38 #define SPDIF\_CLOCKS

**Value:**

```
{
 kCLOCK_Spdif \
}
```

### 4.3.39 #define SRC\_CLOCKS

**Value:**

```
{
 kCLOCK_Src \
}
```

### 4.3.40 #define TMR\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Invalid, kCLOCK_Qtimer1,
 kCLOCK_Qtimer2, kCLOCK_Qtimer3, kCLOCK_Qtimer4,
 kCLOCK_Qtimer5, \
 kCLOCK_Qtimer6, kCLOCK_Qtimer7,
 kCLOCK_Qtimer8 \
}
```

#### 4.3.41 #define TPM\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Invalid, kCLOCK_Tpm1,
 kCLOCK_Tpm2, kCLOCK_Tpm3, kCLOCK_Tpm4,
 kCLOCK_Tpm5, kCLOCK_Tpm6 \
}
```

#### 4.3.42 #define USB\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Usb \
}
```

#### 4.3.43 #define USDHC\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Invalid, kCLOCK_Usdhc1,
 kCLOCK_Usdhc2 \
}
```

#### 4.3.44 #define WDOG\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Invalid, kCLOCK_Wdog1,
 kCLOCK_Wdog2, kCLOCK_Wdog3, kCLOCK_Wdog4,
 kCLOCK_Wdog5 \
}
```

#### 4.3.45 #define XBAR\_CLOCKS

**Value:**

```
{
 \
 kCLOCK_Invalid, kCLOCK_Xbar1,
 kCLOCK_Xbar2, kCLOCK_Xbar3 \
}
```

**4.3.46 #define CLOCK\_GetCoreSysClkFreq CLOCK\_GetCpuClkFreq****4.4 Enumeration Type Documentation****4.4.1 enum clock\_lpcg\_t**

Enumerator

*kCLOCK\_M7* Clock LPCG M7.  
*kCLOCK\_M33* Clock LPCG M33.  
*kCLOCK\_Edgelock* Clock LPCG Edgelock.  
*kCLOCK\_Sim\_Aon* Clock LPCG Sim\_Aon.  
*kCLOCK\_Sim\_Wakeup* Clock LPCG Sim\_Wakeup.  
*kCLOCK\_Sim\_Mega* Clock LPCG Sim\_Mega.  
*kCLOCK\_Sim\_R* Clock LPCG Sim\_R.  
*kCLOCK\_Anadig* Clock LPCG Anadig.  
*kCLOCK\_Dcdc* Clock LPCG Dcdc.  
*kCLOCK\_Src* Clock LPCG Src.  
*kCLOCK\_Ccm* Clock LPCG Ccm.  
*kCLOCK\_Gpc* Clock LPCG Gpc.  
*kCLOCK\_Adc1* Clock LPCG Adc1.  
*kCLOCK\_Adc2* Clock LPCG Adc2.  
*kCLOCK\_Dac* Clock LPCG Dac.  
*kCLOCK\_Acmp1* Clock LPCG Acmp1.  
*kCLOCK\_Acmp2* Clock LPCG Acmp2.  
*kCLOCK\_Acmp3* Clock LPCG Acmp3.  
*kCLOCK\_Acmp4* Clock LPCG Acmp4.  
*kCLOCK\_Wdog1* Clock LPCG Wdog1.  
*kCLOCK\_Wdog2* Clock LPCG Wdog2.  
*kCLOCK\_Wdog3* Clock LPCG Wdog3.  
*kCLOCK\_Wdog4* Clock LPCG Wdog4.  
*kCLOCK\_Wdog5* Clock LPCG Wdog5.  
*kCLOCK\_Ewm0* Clock LPCG Ewm0.  
*kCLOCK\_Sema1* Clock LPCG Sema1.  
*kCLOCK\_Sema2* Clock LPCG Sema2.  
*kCLOCK\_Mu\_A* Clock LPCG Mu\_A.  
*kCLOCK\_Mu\_B* Clock LPCG Mu\_B.  
*kCLOCK\_Edma1* Clock LPCG Edma1.  
*kCLOCK\_Edma2* Clock LPCG Edma2.  
*kCLOCK\_Romcp* Clock LPCG Romcp.  
*kCLOCK\_Ocram1* Clock LPCG Ocram1.  
*kCLOCK\_Ocram2* Clock LPCG Ocram2.  
*kCLOCK\_Flexspi1* Clock LPCG Flexspi1.  
*kCLOCK\_Flexspi2* Clock LPCG Flexspi2.  
*kCLOCK\_Flexspi\_Slv* Clock LPCG Flexspi\_Slv.

*kCLOCK\_Trdc* Clock LPCG Trdc.  
*kCLOCK\_Ocotp* Clock LPCG Octop.  
*kCLOCK\_Semc* Clock LPCG Semc.  
*kCLOCK\_Iee* Clock LPCG Iee.  
*kCLOCK\_Cstrace* Clock LPCG Cstrace.  
*kCLOCK\_Csswo* Clock LPCG Csswo.  
*kCLOCK\_Iomuxc1* Clock LPCG Iomuxc1.  
*kCLOCK\_Iomuxc2* Clock LPCG Iomuxc2.  
*kCLOCK\_Gpio1* Clock LPCG Gpio1.  
*kCLOCK\_Gpio2* Clock LPCG Gpio2.  
*kCLOCK\_Gpio3* Clock LPCG Gpio3.  
*kCLOCK\_Gpio4* Clock LPCG Gpio4.  
*kCLOCK\_Gpio5* Clock LPCG Gpio5.  
*kCLOCK\_Gpio6* Clock LPCG Gpio6.  
*kCLOCK\_Flexio1* Clock LPCG Flexio1.  
*kCLOCK\_Flexio2* Clock LPCG Flexio2.  
*kCLOCK\_Lpit1* Clock LPCG Lpit1.  
*kCLOCK\_Lpit2* Clock LPCG Lpit2.  
*kCLOCK\_Lpit3* Clock LPCG Lpit3.  
*kCLOCK\_Lptmr1* Clock LPCG Lptmr1.  
*kCLOCK\_Lptmr2* Clock LPCG Lptmr2.  
*kCLOCK\_Lptmr3* Clock LPCG Lptmr3.  
*kCLOCK\_Tpm1* Clock LPCG Tpm1.  
*kCLOCK\_Tpm2* Clock LPCG Tpm2.  
*kCLOCK\_Tpm3* Clock LPCG Tpm3.  
*kCLOCK\_Tpm4* Clock LPCG Tpm4.  
*kCLOCK\_Tpm5* Clock LPCG Tpm5.  
*kCLOCK\_Tpm6* Clock LPCG Tpm6.  
*kCLOCK\_Qtimer1* Clock LPCG Qtimer1.  
*kCLOCK\_Qtimer2* Clock LPCG Qtimer2.  
*kCLOCK\_Qtimer3* Clock LPCG Qtimer3.  
*kCLOCK\_Qtimer4* Clock LPCG Qtimer4.  
*kCLOCK\_Qtimer5* Clock LPCG Qtimer5.  
*kCLOCK\_Qtimer6* Clock LPCG Qtimer6.  
*kCLOCK\_Qtimer7* Clock LPCG Qtimer7.  
*kCLOCK\_Qtimer8* Clock LPCG Qtimer8.  
*kCLOCK\_Gpt1* Clock LPCG Gpt1.  
*kCLOCK\_Gpt2* Clock LPCG Gpt2.  
*kCLOCK\_Syscount* Clock LPCG Syscount.  
*kCLOCK\_Can1* Clock LPCG Can1.  
*kCLOCK\_Can2* Clock LPCG Can2.  
*kCLOCK\_Can3* Clock LPCG Can3.  
*kCLOCK\_Lpuart1* Clock LPCG Lpuart1.  
*kCLOCK\_Lpuart2* Clock LPCG Lpuart2.  
*kCLOCK\_Lpuart3* Clock LPCG Lpuart3.

*kCLOCK\_Lpuart4* Clock LPCG Lpuart4.  
*kCLOCK\_Lpuart5* Clock LPCG Lpuart5.  
*kCLOCK\_Lpuart6* Clock LPCG Lpuart6.  
*kCLOCK\_Lpuart7* Clock LPCG Lpuart7.  
*kCLOCK\_Lpuart8* Clock LPCG Lpuart8.  
*kCLOCK\_Lpuart9* Clock LPCG Lpuart9.  
*kCLOCK\_Lpuart10* Clock LPCG Lpuart10.  
*kCLOCK\_Lpuart11* Clock LPCG Lpuart11.  
*kCLOCK\_Lpuart12* Clock LPCG Lpuart12.  
*kCLOCK\_Lpi2c1* Clock LPCG Lpi2c1.  
*kCLOCK\_Lpi2c2* Clock LPCG Lpi2c2.  
*kCLOCK\_Lpi2c3* Clock LPCG Lpi2c3.  
*kCLOCK\_Lpi2c4* Clock LPCG Lpi2c4.  
*kCLOCK\_Lpi2c5* Clock LPCG Lpi2c5.  
*kCLOCK\_Lpi2c6* Clock LPCG Lpi2c6.  
*kCLOCK\_Lpspi1* Clock LPCG Lpspi1.  
*kCLOCK\_Lpspi2* Clock LPCG Lpspi2.  
*kCLOCK\_Lpspi3* Clock LPCG Lpspi3.  
*kCLOCK\_Lpspi4* Clock LPCG Lpspi4.  
*kCLOCK\_Lpspi5* Clock LPCG Lpspi5.  
*kCLOCK\_Lpspi6* Clock LPCG Lpspi6.  
*kCLOCK\_I3c1* Clock LPCG I3c1.  
*kCLOCK\_I3c2* Clock LPCG I3c2.  
*kCLOCK\_Usdhc1* Clock LPCG Usdhc1.  
*kCLOCK\_Usdhc2* Clock LPCG Usdhc2.  
*kCLOCK\_Usb* Clock LPCG Usb.  
*kCLOCK\_Sinc1* Clock LPCG Sinc1.  
*kCLOCK\_Sinc2* Clock LPCG Sinc2.  
*kCLOCK\_Sinc3* Clock LPCG Sinc3.  
*kCLOCK\_Xbar1* Clock LPCG Xbar1.  
*kCLOCK\_Xbar2* Clock LPCG Xbar2.  
*kCLOCK\_Xbar3* Clock LPCG Xbar3.  
*kCLOCK\_Aoi1* Clock LPCG Aoi1.  
*kCLOCK\_Aoi2* Clock LPCG Aoi2.  
*kCLOCK\_Aoi3* Clock LPCG Aoi3.  
*kCLOCK\_Aoi4* Clock LPCG Aoi4.  
*kCLOCK\_Enc1* Clock LPCG Enc1.  
*kCLOCK\_Enc2* Clock LPCG Enc2.  
*kCLOCK\_Enc3* Clock LPCG Enc3.  
*kCLOCK\_Enc4* Clock LPCG Enc4.  
*kCLOCK\_Kpp* Clock LPCG Kpp.  
*kCLOCK\_Pwm1* Clock LPCG Pwm1.  
*kCLOCK\_Pwm2* Clock LPCG Pwm2.  
*kCLOCK\_Pwm3* Clock LPCG Pwm3.  
*kCLOCK\_Pwm4* Clock LPCG Pwm4.

*kCLOCK\_Ecat* Clock LPCG Ecat.  
*kCLOCK\_Netc* Clock LPCG Netc.  
*kCLOCK\_Serdes1* Clock LPCG Serdes1.  
*kCLOCK\_Serdes2* Clock LPCG Serdes2.  
*kCLOCK\_Serdes3* Clock LPCG Serdes3.  
*kCLOCK\_Xcelbusx* Clock LPCG Xcelbusx.  
*kCLOCK\_Xriocu4* Clock LPCG Xriocu4.  
*kCLOCK\_Sptp* Clock LPCG Sptp.  
*kCLOCK\_Mctrl* Clock LPCG Mctrl.  
*kCLOCK\_Sai1* Clock LPCG Sai1.  
*kCLOCK\_Sai2* Clock LPCG Sai2.  
*kCLOCK\_Sai3* Clock LPCG Sai3.  
*kCLOCK\_Sai4* Clock LPCG Sai4.  
*kCLOCK\_Spdif* Clock LPCG Spdif.  
*kCLOCK\_Asrc* Clock LPCG Asrc.  
*kCLOCK\_Pdm* Clock LPCG Mic.  
*kCLOCK\_Vref* Clock LPCG Vref.  
*kCLOCK\_Bist* Clock LPCG Bist.  
*kCLOCK\_Ssi\_W2M7* Clock LPCG Ssi\_W2M7.  
*kCLOCK\_Ssi\_M72W* Clock LPCG Ssi\_M72W.  
*kCLOCK\_Ssi\_W2Ao* Clock LPCG Ssi\_W2Ao.  
*kCLOCK\_Ssi\_Ao2W* Clock LPCG Ssi\_Ao2W.  
*kCLOCK\_Invalid* Invalid value.

#### 4.4.2 enum clock\_name\_t

Enumerator

*kCLOCK\_OscRc24M* 24MHz RC Oscillator.  
*kCLOCK\_OscRc400M* 400MHz RC Oscillator.  
*kCLOCK\_Osc24M* 24MHz Oscillator.  
*kCLOCK\_Osc24MOut* 24MHz Oscillator Out.  
*kCLOCK\_ArmPll* ARM PLL.  
*kCLOCK\_ArmPllOut* ARM PLL Out.  
*kCLOCK\_SysPll2* SYS PLL2.  
*kCLOCK\_SysPll2Out* SYS PLL2 OUT.  
*kCLOCK\_SysPll2Pfd0* SYS PLL2 PFD0.  
*kCLOCK\_SysPll2Pfd1* SYS PLL2 PFD1.  
*kCLOCK\_SysPll2Pfd2* SYS PLL2 PFD2.  
*kCLOCK\_SysPll2Pfd3* SYS PLL2 PFD3.  
*kCLOCK\_SysPll3* SYS PLL3.  
*kCLOCK\_SysPll3Out* SYS PLL3 OUT.  
*kCLOCK\_SysPll3Div2* SYS PLL3 DIV2.  
*kCLOCK\_SysPll3Pfd0* SYS PLL3 PFD0.

*kCLOCK\_SysPll3Pfd1* SYS PLL3 PFD1.  
*kCLOCK\_SysPll3Pfd2* SYS PLL3 PFD2.  
*kCLOCK\_SysPll3Pfd3* SYS PLL3 PFD3.  
*kCLOCK\_SysPll1* SYS PLL1.  
*kCLOCK\_SysPll1Out* SYS PLL1 OUT.  
*kCLOCK\_SysPll1Div2* SYS PLL1 DIV2.  
*kCLOCK\_SysPll1Div5* SYS PLL1 DIV5.  
*kCLOCK\_AudioPll* SYS AUDIO PLL.  
*kCLOCK\_AudioPllOut* SYS AUDIO PLL OUT.  
*kCLOCK\_CpuClk* SYS CPU CLK.  
*kCLOCK\_CoreSysClk* SYS CORE SYS CLK.

#### 4.4.3 enum clock\_root\_t

Enumerator

*kCLOCK\_Root\_M7* CLOCK Root M7.  
*kCLOCK\_Root\_M33* CLOCK Root M33.  
*kCLOCK\_Root\_Edgelock* CLOCK Root Edgelock.  
*kCLOCK\_Root\_Bus\_Aon* CLOCK Root Bus\_Aon.  
*kCLOCK\_Root\_Bus\_Wakeup* CLOCK Root Bus\_Wakeup.  
*kCLOCK\_Root\_Wakeup\_Axi* CLOCK Root Wakeup\_Axi.  
*kCLOCK\_Root\_Swo\_Trace* CLOCK Root Swo\_Trace.  
*kCLOCK\_Root\_M33\_Systick* CLOCK Root M33\_Systick.  
*kCLOCK\_Root\_M7\_Systick* CLOCK Root M7\_Systick.  
*kCLOCK\_Root\_Flexio1* CLOCK Root Flexio1.  
*kCLOCK\_Root\_Flexio2* CLOCK Root Flexio2.  
*kCLOCK\_Root\_Lpit3* CLOCK Root Lpit3.  
*kCLOCK\_Root\_Lptimer1* CLOCK Root Lptimer1.  
*kCLOCK\_Root\_Lptimer2* CLOCK Root Lptimer2.  
*kCLOCK\_Root\_Lptimer3* CLOCK Root Lptimer3.  
*kCLOCK\_Root\_Tpm2* CLOCK Root Tpm2.  
*kCLOCK\_Root\_Tpm4* CLOCK Root Tpm4.  
*kCLOCK\_Root\_Tpm5* CLOCK Root Tpm5.  
*kCLOCK\_Root\_Tpm6* CLOCK Root Tpm6.  
*kCLOCK\_Root\_Gpt1* CLOCK Root Gpt1.  
*kCLOCK\_Root\_Gpt2* CLOCK Root Gpt2.  
*kCLOCK\_Root\_Flexspi1* CLOCK Root Flexspi1.  
*kCLOCK\_Root\_Flexspi2* CLOCK Root Flexspi2.  
*kCLOCK\_Root\_Flexspi\_Slv* CLOCK Root Flexspi\_Slv.  
*kCLOCK\_Root\_Can1* CLOCK Root Can1.  
*kCLOCK\_Root\_Can2* CLOCK Root Can2.  
*kCLOCK\_Root\_Can3* CLOCK Root Can3.  
*kCLOCK\_Root\_Luart0102* CLOCK Root Luart0102.

*kCLOCK\_Root\_Lpuart0304* CLOCK Root Lpuart0304.  
*kCLOCK\_Root\_Lpuart0506* CLOCK Root Lpuart0506.  
*kCLOCK\_Root\_Lpuart0708* CLOCK Root Lpuart0708.  
*kCLOCK\_Root\_Lpuart0910* CLOCK Root Lpuart0910.  
*kCLOCK\_Root\_Lpuart1112* CLOCK Root Lpuart1112.  
*kCLOCK\_Root\_Lpi2c0102* CLOCK Root Lpi2c0102.  
*kCLOCK\_Root\_Lpi2c0304* CLOCK Root Lpi2c0304.  
*kCLOCK\_Root\_Lpi2c0506* CLOCK Root Lpi2c0506.  
*kCLOCK\_Root\_Lpspi0102* CLOCK Root Lpspi0102.  
*kCLOCK\_Root\_Lpspi0304* CLOCK Root Lpspi0304.  
*kCLOCK\_Root\_Lpspi0506* CLOCK Root Lpspi0506.  
*kCLOCK\_Root\_I3c1* CLOCK Root I3c1.  
*kCLOCK\_Root\_I3c2* CLOCK Root I3c2.  
*kCLOCK\_Root\_Usdhc1* CLOCK Root Usdhc1.  
*kCLOCK\_Root\_Usdhc2* CLOCK Root Usdhc2.  
*kCLOCK\_Root\_Semc* CLOCK Root Semc.  
*kCLOCK\_Root\_Adc1* CLOCK Root Adc1.  
*kCLOCK\_Root\_Adc2* CLOCK Root Adc2.  
*kCLOCK\_Root\_Acmp* CLOCK Root Acmp.  
*kCLOCK\_Root\_Ecat* CLOCK Root Ecat.  
*kCLOCK\_Root\_Enet* CLOCK Root Enet.  
*kCLOCK\_Root\_Tmr\_1588* CLOCK Root Tmr\_1588.  
*kCLOCK\_Root\_Netc* CLOCK Root Netc.  
*kCLOCK\_Root\_Mac0* CLOCK Root Mac0.  
*kCLOCK\_Root\_Mac1* CLOCK Root Mac1.  
*kCLOCK\_Root\_Mac2* CLOCK Root Mac2.  
*kCLOCK\_Root\_Mac3* CLOCK Root Mac3.  
*kCLOCK\_Root\_Mac4* CLOCK Root Mac4.  
*kCLOCK\_Root\_Serdes0* CLOCK Root Serdes0.  
*kCLOCK\_Root\_Serdes1* CLOCK Root Serdes1.  
*kCLOCK\_Root\_Serdes2* CLOCK Root Serdes2.  
*kCLOCK\_Root\_Serdes0\_1G* CLOCK Root Serdes0\_1G.  
*kCLOCK\_Root\_Serdes1\_1G* CLOCK Root Serdes1\_1G.  
*kCLOCK\_Root\_Serdes2\_1G* CLOCK Root Serdes2\_1G.  
*kCLOCK\_Root\_Xcelbusx* CLOCK Root Xcelbusx.  
*kCLOCK\_Root\_Xriocu4* CLOCK Root Xriocu4.  
*kCLOCK\_Root\_Mctrl* CLOCK Root Mctrl.  
*kCLOCK\_Root\_Sai1* CLOCK Root Sai1.  
*kCLOCK\_Root\_Sai2* CLOCK Root Sai2.  
*kCLOCK\_Root\_Sai3* CLOCK Root Sai3.  
*kCLOCK\_Root\_Sai4* CLOCK Root Sai4.  
*kCLOCK\_Root\_Spdif* CLOCK Root Spdif.  
*kCLOCK\_Root\_Asrc* CLOCK Root Asrc.  
*kCLOCK\_Root\_Mic* CLOCK Root Mic.  
*kCLOCK\_Root\_Cko1* CLOCK Root Cko1.

**kCLOCK\_Root\_Cko2** CLOCK Root Cko2.

#### 4.4.4 enum clock\_root\_mux\_source\_t

Enumerator

**kCLOCK\_M7\_ClockRoot\_MuxOscRc24M** M7 mux from OscRc24M.  
**kCLOCK\_M7\_ClockRoot\_MuxOscRc400M** M7 mux from OscRc400M.  
**kCLOCK\_M7\_ClockRoot\_MuxArmPllOut** M7 mux from ArmPllOut.  
**kCLOCK\_M7\_ClockRoot\_MuxSysPll3Out** M7 mux from SysPll3Out.  
**kCLOCK\_M33\_ClockRoot\_MuxOscRc24M** M33 mux from OscRc24M.  
**kCLOCK\_M33\_ClockRoot\_MuxOscRc400M** M33 mux from OscRc400M.  
**kCLOCK\_M33\_ClockRoot\_MuxSysPll3Out** M33 mux from SysPll3Out.  
**kCLOCK\_M33\_ClockRoot\_MuxArmPllOut** M33 mux from ArmPllOut.  
**kCLOCK\_EDGELOCK\_ClockRoot\_MuxOscRc24M** EDGELOCK mux from OscRc24M.  
**kCLOCK\_EDGELOCK\_ClockRoot\_MuxOscRc400M** EDGELOCK mux from OscRc400M.  
**kCLOCK\_EDGELOCK\_ClockRoot\_MuxSysPll1Out** EDGELOCK mux from SysPll1Out.  
**kCLOCK\_EDGELOCK\_ClockRoot\_MuxSysPll2Pfd1** EDGELOCK mux from SysPll2Pfd1.  
**kCLOCK\_BUS\_AON\_ClockRoot\_MuxOscRc24M** BUS\_AON mux from OscRc24M.  
**kCLOCK\_BUS\_AON\_ClockRoot\_MuxOscRc400M** BUS\_AON mux from OscRc400M.  
**kCLOCK\_BUS\_AON\_ClockRoot\_MuxSysPll2Out** BUS\_AON mux from SysPll2Out.  
**kCLOCK\_BUS\_AON\_ClockRoot\_MuxSysPll3Pfd2** BUS\_AON mux from SysPll3Pfd2.  
**kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxOscRc24M** BUS\_WAKEUP mux from OscRc24M.  
**kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxOscRc400M** BUS\_WAKEUP mux from OscRc400M.  
**kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxSysPll2Out** BUS\_WAKEUP mux from SysPll2Out.  
**kCLOCK\_BUS\_WAKEUP\_ClockRoot\_MuxSysPll3Pfd1** BUS\_WAKEUP mux from SysPll3Pfd1.

**kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxOscRc24M** WAKEUP\_AXI mux from OscRc24M.  
**kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxOscRc400M** WAKEUP\_AXI mux from OscRc400M.  
**kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxSysPll3Out** WAKEUP\_AXI mux from SysPll3Out.  
**kCLOCK\_WAKEUP\_AXI\_ClockRoot\_MuxSysPll2Pfd1** WAKEUP\_AXI mux from SysPll2Pfd1.  
**kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxOscRc24M** SWO\_TRACE mux from OscRc24M.  
**kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxOscRc400M** SWO\_TRACE mux from OscRc400M.  
**kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxSysPll3Div2** SWO\_TRACE mux from SysPll3Div2.  
**kCLOCK\_SWO\_TRACE\_ClockRoot\_MuxSysPll1Div5** SWO\_TRACE mux from SysPll1Div5.  
**kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxOscRc24M** M33\_SYSTICK mux from OscRc24M.  
**kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxOscRc400M** M33\_SYSTICK mux from OscRc400M.  
**kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxOsc24MOut** M33\_SYSTICK mux from Osc24MOut.  
**kCLOCK\_M33\_SYSTICK\_ClockRoot\_MuxSysPll3Div2** M33\_SYSTICK mux from SysPll3Div2.  
**kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxOscRc24M** M7\_SYSTICK mux from OscRc24M.  
**kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxOscRc400M** M7\_SYSTICK mux from OscRc400M.  
**kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxOsc24MOut** M7\_SYSTICK mux from Osc24MOut.  
**kCLOCK\_M7\_SYSTICK\_ClockRoot\_MuxSysPll3Div2** M7\_SYSTICK mux from SysPll3Div2.

**kCLOCK\_FLEXIO1\_ClockRoot\_MuxOscRc24M** FLEXIO1 mux from OscRc24M.  
**kCLOCK\_FLEXIO1\_ClockRoot\_MuxOscRc400M** FLEXIO1 mux from OscRc400M.  
**kCLOCK\_FLEXIO1\_ClockRoot\_MuxSysPll3Div2** FLEXIO1 mux from SysPll3Div2.  
**kCLOCK\_FLEXIO1\_ClockRoot\_MuxSysPll1Div5** FLEXIO1 mux from SysPll1Div5.  
**kCLOCK\_FLEXIO2\_ClockRoot\_MuxOscRc24M** FLEXIO2 mux from OscRc24M.  
**kCLOCK\_FLEXIO2\_ClockRoot\_MuxOscRc400M** FLEXIO2 mux from OscRc400M.  
**kCLOCK\_FLEXIO2\_ClockRoot\_MuxSysPll3Div2** FLEXIO2 mux from SysPll3Div2.  
**kCLOCK\_FLEXIO2\_ClockRoot\_MuxSysPll1Div5** FLEXIO2 mux from SysPll1Div5.  
**kCLOCK\_LPIT3\_ClockRoot\_MuxOscRc24M** LPIT3 mux from OscRc24M.  
**kCLOCK\_LPIT3\_ClockRoot\_MuxOscRc400M** LPIT3 mux from OscRc400M.  
**kCLOCK\_LPIT3\_ClockRoot\_MuxSysPll3Div2** LPIT3 mux from SysPll3Div2.  
**kCLOCK\_LPIT3\_ClockRoot\_MuxSysPll2Pfd3** LPIT3 mux from SysPll2Pfd3.  
**kCLOCK\_LPTIMER1\_ClockRoot\_MuxOscRc24M** LPTIMER1 mux from OscRc24M.  
**kCLOCK\_LPTIMER1\_ClockRoot\_MuxOscRc400M** LPTIMER1 mux from OscRc400M.  
**kCLOCK\_LPTIMER1\_ClockRoot\_MuxSysPll3Div2** LPTIMER1 mux from SysPll3Div2.  
**kCLOCK\_LPTIMER1\_ClockRoot\_MuxSysPll2Pfd3** LPTIMER1 mux from SysPll2Pfd3.  
**kCLOCK\_LPTIMER2\_ClockRoot\_MuxOscRc24M** LPTIMER2 mux from OscRc24M.  
**kCLOCK\_LPTIMER2\_ClockRoot\_MuxOscRc400M** LPTIMER2 mux from OscRc400M.  
**kCLOCK\_LPTIMER2\_ClockRoot\_MuxSysPll3Div2** LPTIMER2 mux from SysPll3Div2.  
**kCLOCK\_LPTIMER2\_ClockRoot\_MuxSysPll2Pfd3** LPTIMER2 mux from SysPll2Pfd3.  
**kCLOCK\_LPTIMER3\_ClockRoot\_MuxOscRc24M** LPTIMER3 mux from OscRc24M.  
**kCLOCK\_LPTIMER3\_ClockRoot\_MuxOscRc400M** LPTIMER3 mux from OscRc400M.  
**kCLOCK\_LPTIMER3\_ClockRoot\_MuxSysPll3Div2** LPTIMER3 mux from SysPll3Div2.  
**kCLOCK\_LPTIMER3\_ClockRoot\_MuxSysPll2Pfd3** LPTIMER3 mux from SysPll2Pfd3.  
**kCLOCK TPM2\_ClockRoot\_MuxOscRc24M** TPM2 mux from OscRc24M.  
**kCLOCK TPM2\_ClockRoot\_MuxOscRc400M** TPM2 mux from OscRc400M.  
**kCLOCK TPM2\_ClockRoot\_MuxSysPll3Div2** TPM2 mux from SysPll3Div2.  
**kCLOCK TPM2\_ClockRoot\_MuxSysPll2Pfd3** TPM2 mux from SysPll2Pfd3.  
**kCLOCK TPM4\_ClockRoot\_MuxOscRc24M** TPM4 mux from OscRc24M.  
**kCLOCK TPM4\_ClockRoot\_MuxOscRc400M** TPM4 mux from OscRc400M.  
**kCLOCK TPM4\_ClockRoot\_MuxSysPll3Div2** TPM4 mux from SysPll3Div2.  
**kCLOCK TPM4\_ClockRoot\_MuxSysPll2Pfd3** TPM4 mux from SysPll2Pfd3.  
**kCLOCK TPM5\_ClockRoot\_MuxOscRc24M** TPM5 mux from OscRc24M.  
**kCLOCK TPM5\_ClockRoot\_MuxOscRc400M** TPM5 mux from OscRc400M.  
**kCLOCK TPM5\_ClockRoot\_MuxSysPll3Div2** TPM5 mux from SysPll3Div2.  
**kCLOCK TPM5\_ClockRoot\_MuxSysPll2Pfd3** TPM5 mux from SysPll2Pfd3.  
**kCLOCK TPM6\_ClockRoot\_MuxOscRc24M** TPM6 mux from OscRc24M.  
**kCLOCK TPM6\_ClockRoot\_MuxOscRc400M** TPM6 mux from OscRc400M.  
**kCLOCK TPM6\_ClockRoot\_MuxSysPll3Div2** TPM6 mux from SysPll3Div2.  
**kCLOCK TPM6\_ClockRoot\_MuxSysPll2Pfd3** TPM6 mux from SysPll2Pfd3.  
**kCLOCK\_GPT1\_ClockRoot\_MuxOscRc24M** GPT1 mux from OscRc24M.  
**kCLOCK\_GPT1\_ClockRoot\_MuxOscRc400M** GPT1 mux from OscRc400M.  
**kCLOCK\_GPT1\_ClockRoot\_MuxSysPll3Div2** GPT1 mux from SysPll3Div2.  
**kCLOCK\_GPT1\_ClockRoot\_MuxSysPll2Pfd3** GPT1 mux from SysPll2Pfd3.  
**kCLOCK\_GPT2\_ClockRoot\_MuxOscRc24M** GPT2 mux from OscRc24M.

**kCLOCK\_GPT2\_ClockRoot\_MuxOscRc400M** GPT2 mux from OscRc400M.  
**kCLOCK\_GPT2\_ClockRoot\_MuxSysPll3Div2** GPT2 mux from SysPll3Div2.  
**kCLOCK\_GPT2\_ClockRoot\_MuxSysPll2Pfd3** GPT2 mux from SysPll2Pfd3.  
**kCLOCK\_FLEXSPII1\_ClockRoot\_MuxOscRc24M** FLEXSPII1 mux from OscRc24M.  
**kCLOCK\_FLEXSPII1\_ClockRoot\_MuxOscRc400M** FLEXSPII1 mux from OscRc400M.  
**kCLOCK\_FLEXSPII1\_ClockRoot\_MuxSysPll3Pfd0** FLEXSPII1 mux from SysPll3Pfd0.  
**kCLOCK\_FLEXSPII1\_ClockRoot\_MuxSysPll2Pfd0** FLEXSPII1 mux from SysPll2Pfd0.  
**kCLOCK\_FLEXSPII2\_ClockRoot\_MuxOscRc24M** FLEXSPII2 mux from OscRc24M.  
**kCLOCK\_FLEXSPII2\_ClockRoot\_MuxOscRc400M** FLEXSPII2 mux from OscRc400M.  
**kCLOCK\_FLEXSPII2\_ClockRoot\_MuxSysPll3Pfd2** FLEXSPII2 mux from SysPll3Pfd2.  
**kCLOCK\_FLEXSPII2\_ClockRoot\_MuxSysPll2Pfd1** FLEXSPII2 mux from SysPll2Pfd1.  
**kCLOCK\_FLEXSPI\_SLV\_ClockRoot\_MuxOscRc24M** FLEXSPI\_SLV mux from OscRc24M.  
**kCLOCK\_FLEXSPI\_SLV\_ClockRoot\_MuxOscRc400M** FLEXSPI\_SLV mux from OscRc400M.  
**kCLOCK\_FLEXSPI\_SLV\_ClockRoot\_MuxSysPll2Out** FLEXSPI\_SLV mux from SysPll2Out.  
**kCLOCK\_FLEXSPI\_SLV\_ClockRoot\_MuxSysPll1Out** FLEXSPI\_SLV mux from SysPll1Out.  
**kCLOCK\_CAN1\_ClockRoot\_MuxOscRc24M** CAN1 mux from OscRc24M.  
**kCLOCK\_CAN1\_ClockRoot\_MuxOscRc400M** CAN1 mux from OscRc400M.  
**kCLOCK\_CAN1\_ClockRoot\_MuxSysPll3Out** CAN1 mux from SysPll3Out.  
**kCLOCK\_CAN1\_ClockRoot\_MuxOsc24MOut** CAN1 mux from Osc24MOut.  
**kCLOCK\_CAN2\_ClockRoot\_MuxOscRc24M** CAN2 mux from OscRc24M.  
**kCLOCK\_CAN2\_ClockRoot\_MuxOscRc400M** CAN2 mux from OscRc400M.  
**kCLOCK\_CAN2\_ClockRoot\_MuxSysPll3Out** CAN2 mux from SysPll3Out.  
**kCLOCK\_CAN2\_ClockRoot\_MuxOsc24MOut** CAN2 mux from Osc24MOut.  
**kCLOCK\_CAN3\_ClockRoot\_MuxOscRc24M** CAN3 mux from OscRc24M.  
**kCLOCK\_CAN3\_ClockRoot\_MuxOscRc400M** CAN3 mux from OscRc400M.  
**kCLOCK\_CAN3\_ClockRoot\_MuxSysPll3Out** CAN3 mux from SysPll3Out.  
**kCLOCK\_CAN3\_ClockRoot\_MuxOsc24MOut** CAN3 mux from Osc24MOut.  
**kCLOCK\_LPUART0102\_ClockRoot\_MuxOscRc24M** LPUART0102 mux from OscRc24M.  
**kCLOCK\_LPUART0102\_ClockRoot\_MuxOscRc400M** LPUART0102 mux from OscRc400M.  
**kCLOCK\_LPUART0102\_ClockRoot\_MuxSysPll3Div2** LPUART0102 mux from SysPll3Div2.  
**kCLOCK\_LPUART0102\_ClockRoot\_MuxSysPll2Pfd3** LPUART0102 mux from SysPll2Pfd3.  
**kCLOCK\_LPUART0304\_ClockRoot\_MuxOscRc24M** LPUART0304 mux from OscRc24M.  
**kCLOCK\_LPUART0304\_ClockRoot\_MuxOscRc400M** LPUART0304 mux from OscRc400M.  
**kCLOCK\_LPUART0304\_ClockRoot\_MuxSysPll3Div2** LPUART0304 mux from SysPll3Div2.  
**kCLOCK\_LPUART0304\_ClockRoot\_MuxSysPll2Pfd3** LPUART0304 mux from SysPll2Pfd3.  
**kCLOCK\_LPUART0506\_ClockRoot\_MuxOscRc24M** LPUART0506 mux from OscRc24M.  
**kCLOCK\_LPUART0506\_ClockRoot\_MuxOscRc400M** LPUART0506 mux from OscRc400M.  
**kCLOCK\_LPUART0506\_ClockRoot\_MuxSysPll3Div2** LPUART0506 mux from SysPll3Div2.  
**kCLOCK\_LPUART0506\_ClockRoot\_MuxSysPll2Pfd3** LPUART0506 mux from SysPll2Pfd3.  
**kCLOCK\_LPUART0708\_ClockRoot\_MuxOscRc24M** LPUART0708 mux from OscRc24M.  
**kCLOCK\_LPUART0708\_ClockRoot\_MuxOscRc400M** LPUART0708 mux from OscRc400M.  
**kCLOCK\_LPUART0708\_ClockRoot\_MuxSysPll3Div2** LPUART0708 mux from SysPll3Div2.  
**kCLOCK\_LPUART0708\_ClockRoot\_MuxSysPll2Pfd3** LPUART0708 mux from SysPll2Pfd3.  
**kCLOCK\_LPUART0910\_ClockRoot\_MuxOscRc24M** LPUART0910 mux from OscRc24M.  
**kCLOCK\_LPUART0910\_ClockRoot\_MuxOscRc400M** LPUART0910 mux from OscRc400M.

**kCLOCK\_LPUART0910\_ClockRoot\_MuxSysPll3Div2** LPUART0910 mux from SysPll3Div2.  
**kCLOCK\_LPUART0910\_ClockRoot\_MuxSysPll2Pfd3** LPUART0910 mux from SysPll2Pfd3.  
**kCLOCK\_LPUART1112\_ClockRoot\_MuxOscRc24M** LPUART1112 mux from OscRc24M.  
**kCLOCK\_LPUART1112\_ClockRoot\_MuxOscRc400M** LPUART1112 mux from OscRc400M.  
**kCLOCK\_LPUART1112\_ClockRoot\_MuxSysPll3Div2** LPUART1112 mux from SysPll3Div2.  
**kCLOCK\_LPUART1112\_ClockRoot\_MuxSysPll2Pfd3** LPUART1112 mux from SysPll2Pfd3.  
**kCLOCK\_LPI2C0102\_ClockRoot\_MuxOscRc24M** LPI2C0102 mux from OscRc24M.  
**kCLOCK\_LPI2C0102\_ClockRoot\_MuxOscRc400M** LPI2C0102 mux from OscRc400M.  
**kCLOCK\_LPI2C0102\_ClockRoot\_MuxSysPll3Div2** LPI2C0102 mux from SysPll3Div2.  
**kCLOCK\_LPI2C0102\_ClockRoot\_MuxSysPll2Pfd3** LPI2C0102 mux from SysPll2Pfd3.  
**kCLOCK\_LPI2C0304\_ClockRoot\_MuxOscRc24M** LPI2C0304 mux from OscRc24M.  
**kCLOCK\_LPI2C0304\_ClockRoot\_MuxOscRc400M** LPI2C0304 mux from OscRc400M.  
**kCLOCK\_LPI2C0304\_ClockRoot\_MuxSysPll3Div2** LPI2C0304 mux from SysPll3Div2.  
**kCLOCK\_LPI2C0304\_ClockRoot\_MuxSysPll2Pfd3** LPI2C0304 mux from SysPll2Pfd3.  
**kCLOCK\_LPI2C0506\_ClockRoot\_MuxOscRc24M** LPI2C0506 mux from OscRc24M.  
**kCLOCK\_LPI2C0506\_ClockRoot\_MuxOscRc400M** LPI2C0506 mux from OscRc400M.  
**kCLOCK\_LPI2C0506\_ClockRoot\_MuxSysPll3Div2** LPI2C0506 mux from SysPll3Div2.  
**kCLOCK\_LPI2C0506\_ClockRoot\_MuxSysPll2Pfd3** LPI2C0506 mux from SysPll2Pfd3.  
**kCLOCK\_LP SPI0102\_ClockRoot\_MuxOscRc24M** LP SPI0102 mux from OscRc24M.  
**kCLOCK\_LP SPI0102\_ClockRoot\_MuxOscRc400M** LP SPI0102 mux from OscRc400M.  
**kCLOCK\_LP SPI0102\_ClockRoot\_MuxSysPll3Pfd1** LP SPI0102 mux from SysPll3Pfd1.  
**kCLOCK\_LP SPI0102\_ClockRoot\_MuxSysPll2Out** LP SPI0102 mux from SysPll2Out.  
**kCLOCK\_LP SPI0304\_ClockRoot\_MuxOscRc24M** LP SPI0304 mux from OscRc24M.  
**kCLOCK\_LP SPI0304\_ClockRoot\_MuxOscRc400M** LP SPI0304 mux from OscRc400M.  
**kCLOCK\_LP SPI0304\_ClockRoot\_MuxSysPll3Pfd1** LP SPI0304 mux from SysPll3Pfd1.  
**kCLOCK\_LP SPI0304\_ClockRoot\_MuxSysPll2Out** LP SPI0304 mux from SysPll2Out.  
**kCLOCK\_LP SPI0506\_ClockRoot\_MuxOscRc24M** LP SPI0506 mux from OscRc24M.  
**kCLOCK\_LP SPI0506\_ClockRoot\_MuxOscRc400M** LP SPI0506 mux from OscRc400M.  
**kCLOCK\_LP SPI0506\_ClockRoot\_MuxSysPll3Pfd1** LP SPI0506 mux from SysPll3Pfd1.  
**kCLOCK\_LP SPI0506\_ClockRoot\_MuxSysPll2Out** LP SPI0506 mux from SysPll2Out.  
**kCLOCK\_I3C1\_ClockRoot\_MuxOscRc24M** I3C1 mux from OscRc24M.  
**kCLOCK\_I3C1\_ClockRoot\_MuxOscRc400M** I3C1 mux from OscRc400M.  
**kCLOCK\_I3C1\_ClockRoot\_MuxSysPll3Div2** I3C1 mux from SysPll3Div2.  
**kCLOCK\_I3C1\_ClockRoot\_MuxSysPll2Pfd3** I3C1 mux from SysPll2Pfd3.  
**kCLOCK\_I3C2\_ClockRoot\_MuxOscRc24M** I3C2 mux from OscRc24M.  
**kCLOCK\_I3C2\_ClockRoot\_MuxOscRc400M** I3C2 mux from OscRc400M.  
**kCLOCK\_I3C2\_ClockRoot\_MuxSysPll3Div2** I3C2 mux from SysPll3Div2.  
**kCLOCK\_I3C2\_ClockRoot\_MuxSysPll2Pfd3** I3C2 mux from SysPll2Pfd3.  
**kCLOCK\_USDHC1\_ClockRoot\_MuxOscRc24M** USDHC1 mux from OscRc24M.  
**kCLOCK\_USDHC1\_ClockRoot\_MuxOscRc400M** USDHC1 mux from OscRc400M.  
**kCLOCK\_USDHC1\_ClockRoot\_MuxSysPll2Pfd2** USDHC1 mux from SysPll2Pfd2.  
**kCLOCK\_USDHC1\_ClockRoot\_MuxSysPll1Div5** USDHC1 mux from SysPll1Div5.  
**kCLOCK\_USDHC2\_ClockRoot\_MuxOscRc24M** USDHC2 mux from OscRc24M.  
**kCLOCK\_USDHC2\_ClockRoot\_MuxOscRc400M** USDHC2 mux from OscRc400M.  
**kCLOCK\_USDHC2\_ClockRoot\_MuxSysPll2Pfd2** USDHC2 mux from SysPll2Pfd2.

*kCLOCK\_USDHC2\_ClockRoot\_MuxSysPll1Div5* USDHC2 mux from SysPll1Div5.  
*kCLOCK\_SEMC\_ClockRoot\_MuxOscRc24M* SEMC mux from OscRc24M.  
*kCLOCK\_SEMC\_ClockRoot\_MuxOscRc400M* SEMC mux from OscRc400M.  
*kCLOCK\_SEMC\_ClockRoot\_MuxSysPll1Out* SEMC mux from SysPll1Out.  
*kCLOCK\_SEMC\_ClockRoot\_MuxSysPll2Pfd0* SEMC mux from SysPll2Pfd0.  
*kCLOCK\_ADC1\_ClockRoot\_MuxOscRc24M* ADC1 mux from OscRc24M.  
*kCLOCK\_ADC1\_ClockRoot\_MuxOscRc400M* ADC1 mux from OscRc400M.  
*kCLOCK\_ADC1\_ClockRoot\_MuxSysPll3Div2* ADC1 mux from SysPll3Div2.  
*kCLOCK\_ADC1\_ClockRoot\_MuxSysPll2Pfd3* ADC1 mux from SysPll2Pfd3.  
*kCLOCK\_ADC2\_ClockRoot\_MuxOscRc24M* ADC2 mux from OscRc24M.  
*kCLOCK\_ADC2\_ClockRoot\_MuxOscRc400M* ADC2 mux from OscRc400M.  
*kCLOCK\_ADC2\_ClockRoot\_MuxSysPll3Div2* ADC2 mux from SysPll3Div2.  
*kCLOCK\_ADC2\_ClockRoot\_MuxSysPll2Pfd3* ADC2 mux from SysPll2Pfd3.  
*kCLOCK\_ACMP\_ClockRoot\_MuxOscRc24M* ACMP mux from OscRc24M.  
*kCLOCK\_ACMP\_ClockRoot\_MuxOscRc400M* ACMP mux from OscRc400M.  
*kCLOCK\_ACMP\_ClockRoot\_MuxSysPll3Out* ACMP mux from SysPll3Out.  
*kCLOCK\_ACMP\_ClockRoot\_MuxSysPll2Pfd3* ACMP mux from SysPll2Pfd3.  
*kCLOCK\_ECAT\_ClockRoot\_MuxOscRc24M* ECAT mux from OscRc24M.  
*kCLOCK\_ECAT\_ClockRoot\_MuxOscRc400M* ECAT mux from OscRc400M.  
*kCLOCK\_ECAT\_ClockRoot\_MuxSysPll1Div2* ECAT mux from SysPll1Div2.  
*kCLOCK\_ECAT\_ClockRoot\_MuxSysPll1Div5* ECAT mux from SysPll1Div5.  
*kCLOCK\_ENET\_ClockRoot\_MuxOscRc24M* ENET mux from OscRc24M.  
*kCLOCK\_ENET\_ClockRoot\_MuxOscRc400M* ENET mux from OscRc400M.  
*kCLOCK\_ENET\_ClockRoot\_MuxSysPll1Div2* ENET mux from SysPll1Div2.  
*kCLOCK\_ENET\_ClockRoot\_MuxSysPll1Div5* ENET mux from SysPll1Div5.  
*kCLOCK\_TMR\_1588\_ClockRoot\_MuxOscRc24M* TMR\_1588 mux from OscRc24M.  
*kCLOCK\_TMR\_1588\_ClockRoot\_MuxOscRc400M* TMR\_1588 mux from OscRc400M.  
*kCLOCK\_TMR\_1588\_ClockRoot\_MuxSysPll3Out* TMR\_1588 mux from SysPll3Out.  
*kCLOCK\_TMR\_1588\_ClockRoot\_MuxSysPll2Pfd3* TMR\_1588 mux from SysPll2Pfd3.  
*kCLOCK\_NETC\_ClockRoot\_MuxOscRc24M* NETC mux from OscRc24M.  
*kCLOCK\_NETC\_ClockRoot\_MuxOscRc400M* NETC mux from OscRc400M.  
*kCLOCK\_NETC\_ClockRoot\_MuxSysPll3Pfd3* NETC mux from SysPll3Pfd3.  
*kCLOCK\_NETC\_ClockRoot\_MuxSysPll2Pfd1* NETC mux from SysPll2Pfd1.  
*kCLOCK\_MAC0\_ClockRoot\_MuxOscRc24M* MAC0 mux from OscRc24M.  
*kCLOCK\_MAC0\_ClockRoot\_MuxOscRc400M* MAC0 mux from OscRc400M.  
*kCLOCK\_MAC0\_ClockRoot\_MuxSysPll1Div2* MAC0 mux from SysPll1Div2.  
*kCLOCK\_MAC0\_ClockRoot\_MuxSysPll1Div5* MAC0 mux from SysPll1Div5.  
*kCLOCK\_MAC1\_ClockRoot\_MuxOscRc24M* MAC1 mux from OscRc24M.  
*kCLOCK\_MAC1\_ClockRoot\_MuxOscRc400M* MAC1 mux from OscRc400M.  
*kCLOCK\_MAC1\_ClockRoot\_MuxSysPll1Div2* MAC1 mux from SysPll1Div2.  
*kCLOCK\_MAC1\_ClockRoot\_MuxSysPll1Div5* MAC1 mux from SysPll1Div5.  
*kCLOCK\_MAC2\_ClockRoot\_MuxOscRc24M* MAC2 mux from OscRc24M.  
*kCLOCK\_MAC2\_ClockRoot\_MuxOscRc400M* MAC2 mux from OscRc400M.  
*kCLOCK\_MAC2\_ClockRoot\_MuxSysPll1Div2* MAC2 mux from SysPll1Div2.  
*kCLOCK\_MAC2\_ClockRoot\_MuxSysPll1Div5* MAC2 mux from SysPll1Div5.

***kCLOCK\_MAC3\_ClockRoot\_MuxOscRc24M*** MAC3 mux from OscRc24M.  
***kCLOCK\_MAC3\_ClockRoot\_MuxOscRc400M*** MAC3 mux from OscRc400M.  
***kCLOCK\_MAC3\_ClockRoot\_MuxSysPll1Div2*** MAC3 mux from SysPll1Div2.  
***kCLOCK\_MAC3\_ClockRoot\_MuxSysPll1Div5*** MAC3 mux from SysPll1Div5.  
***kCLOCK\_MAC4\_ClockRoot\_MuxOscRc24M*** MAC4 mux from OscRc24M.  
***kCLOCK\_MAC4\_ClockRoot\_MuxOscRc400M*** MAC4 mux from OscRc400M.  
***kCLOCK\_MAC4\_ClockRoot\_MuxSysPll1Div2*** MAC4 mux from SysPll1Div2.  
***kCLOCK\_MAC4\_ClockRoot\_MuxSysPll1Div5*** MAC4 mux from SysPll1Div5.  
***kCLOCK\_SERDES0\_ClockRoot\_MuxOscRc24M*** SERDES0 mux from OscRc24M.  
***kCLOCK\_SERDES0\_ClockRoot\_MuxOscRc400M*** SERDES0 mux from OscRc400M.  
***kCLOCK\_SERDES0\_ClockRoot\_MuxSysPll1Div2*** SERDES0 mux from SysPll1Div2.  
***kCLOCK\_SERDES0\_ClockRoot\_MuxSysPll1Div5*** SERDES0 mux from SysPll1Div5.  
***kCLOCK\_SERDES1\_ClockRoot\_MuxOscRc24M*** SERDES1 mux from OscRc24M.  
***kCLOCK\_SERDES1\_ClockRoot\_MuxOscRc400M*** SERDES1 mux from OscRc400M.  
***kCLOCK\_SERDES1\_ClockRoot\_MuxSysPll1Div2*** SERDES1 mux from SysPll1Div2.  
***kCLOCK\_SERDES1\_ClockRoot\_MuxSysPll1Div5*** SERDES1 mux from SysPll1Div5.  
***kCLOCK\_SERDES2\_ClockRoot\_MuxOscRc24M*** SERDES2 mux from OscRc24M.  
***kCLOCK\_SERDES2\_ClockRoot\_MuxOscRc400M*** SERDES2 mux from OscRc400M.  
***kCLOCK\_SERDES2\_ClockRoot\_MuxSysPll1Div2*** SERDES2 mux from SysPll1Div2.  
***kCLOCK\_SERDES2\_ClockRoot\_MuxSysPll1Div5*** SERDES2 mux from SysPll1Div5.  
***kCLOCK\_SERDES0\_1G\_ClockRoot\_MuxOscRc24M*** SERDES0\_1G mux from OscRc24M.  
***kCLOCK\_SERDES0\_1G\_ClockRoot\_MuxOscRc400M*** SERDES0\_1G mux from OscRc400M.  
***kCLOCK\_SERDES0\_1G\_ClockRoot\_MuxSysPll1Out*** SERDES0\_1G mux from SysPll1Out.  
***kCLOCK\_SERDES0\_1G\_ClockRoot\_MuxAudioPllOut*** SERDES0\_1G mux from AudioPllOut.  
***kCLOCK\_SERDES1\_1G\_ClockRoot\_MuxOscRc24M*** SERDES1\_1G mux from OscRc24M.  
***kCLOCK\_SERDES1\_1G\_ClockRoot\_MuxOscRc400M*** SERDES1\_1G mux from OscRc400M.  
***kCLOCK\_SERDES1\_1G\_ClockRoot\_MuxSysPll1Out*** SERDES1\_1G mux from SysPll1Out.  
***kCLOCK\_SERDES1\_1G\_ClockRoot\_MuxAudioPllOut*** SERDES1\_1G mux from AudioPllOut.  
***kCLOCK\_SERDES2\_1G\_ClockRoot\_MuxOscRc24M*** SERDES2\_1G mux from OscRc24M.  
***kCLOCK\_SERDES2\_1G\_ClockRoot\_MuxOscRc400M*** SERDES2\_1G mux from OscRc400M.  
***kCLOCK\_SERDES2\_1G\_ClockRoot\_MuxSysPll1Out*** SERDES2\_1G mux from SysPll1Out.  
***kCLOCK\_SERDES2\_1G\_ClockRoot\_MuxAudioPllOut*** SERDES2\_1G mux from AudioPllOut.  
***kCLOCK\_XCELBUSX\_ClockRoot\_MuxOscRc24M*** XCELBUSX mux from OscRc24M.  
***kCLOCK\_XCELBUSX\_ClockRoot\_MuxOscRc400M*** XCELBUSX mux from OscRc400M.  
***kCLOCK\_XCELBUSX\_ClockRoot\_MuxSysPll3Out*** XCELBUSX mux from SysPll3Out.  
***kCLOCK\_XCELBUSX\_ClockRoot\_MuxSysPll3Pfd1*** XCELBUSX mux from SysPll3Pfd1.  
***kCLOCK\_XRIOCU4\_ClockRoot\_MuxOscRc24M*** XRIOCU4 mux from OscRc24M.  
***kCLOCK\_XRIOCU4\_ClockRoot\_MuxOscRc400M*** XRIOCU4 mux from OscRc400M.  
***kCLOCK\_XRIOCU4\_ClockRoot\_MuxOsc24MOut*** XRIOCU4 mux from Osc24MOut.  
***kCLOCK\_XRIOCU4\_ClockRoot\_MuxSysPll3Div2*** XRIOCU4 mux from SysPll3Div2.  
***kCLOCK\_MCTRL\_ClockRoot\_MuxOscRc24M*** MCTRL mux from OscRc24M.  
***kCLOCK\_MCTRL\_ClockRoot\_MuxOscRc400M*** MCTRL mux from OscRc400M.  
***kCLOCK\_MCTRL\_ClockRoot\_MuxSysPll1Div5*** MCTRL mux from SysPll1Div5.  
***kCLOCK\_MCTRL\_ClockRoot\_MuxAudioPllOut*** MCTRL mux from AudioPllOut.  
***kCLOCK\_SAI1\_ClockRoot\_MuxOscRc24M*** SAI1 mux from OscRc24M.

*kCLOCK\_SAI1\_ClockRoot\_MuxOscRc400M* SAI1 mux from OscRc400M.  
*kCLOCK\_SAI1\_ClockRoot\_MuxAudioPllOut* SAI1 mux from AudioPllOut.  
*kCLOCK\_SAI1\_ClockRoot\_MuxSysPll3Pfd2* SAI1 mux from SysPll3Pfd2.  
*kCLOCK\_SAI2\_ClockRoot\_MuxOscRc24M* SAI2 mux from OscRc24M.  
*kCLOCK\_SAI2\_ClockRoot\_MuxOscRc400M* SAI2 mux from OscRc400M.  
*kCLOCK\_SAI2\_ClockRoot\_MuxAudioPllOut* SAI2 mux from AudioPllOut.  
*kCLOCK\_SAI2\_ClockRoot\_MuxSysPll3Pfd2* SAI2 mux from SysPll3Pfd2.  
*kCLOCK\_SAI3\_ClockRoot\_MuxOscRc24M* SAI3 mux from OscRc24M.  
*kCLOCK\_SAI3\_ClockRoot\_MuxOscRc400M* SAI3 mux from OscRc400M.  
*kCLOCK\_SAI3\_ClockRoot\_MuxAudioPllOut* SAI3 mux from AudioPllOut.  
*kCLOCK\_SAI3\_ClockRoot\_MuxSysPll3Pfd2* SAI3 mux from SysPll3Pfd2.  
*kCLOCK\_SAI4\_ClockRoot\_MuxOscRc24M* SAI4 mux from OscRc24M.  
*kCLOCK\_SAI4\_ClockRoot\_MuxOscRc400M* SAI4 mux from OscRc400M.  
*kCLOCK\_SAI4\_ClockRoot\_MuxAudioPllOut* SAI4 mux from AudioPllOut.  
*kCLOCK\_SAI4\_ClockRoot\_MuxSysPll3Pfd2* SAI4 mux from SysPll3Pfd2.  
*kCLOCK\_SPDIF\_ClockRoot\_MuxOscRc24M* SPDIF mux from OscRc24M.  
*kCLOCK\_SPDIF\_ClockRoot\_MuxOscRc400M* SPDIF mux from OscRc400M.  
*kCLOCK\_SPDIF\_ClockRoot\_MuxAudioPllOut* SPDIF mux from AudioPllOut.  
*kCLOCK\_SPDIF\_ClockRoot\_MuxSysPll3Pfd2* SPDIF mux from SysPll3Pfd2.  
*kCLOCK\_ASRC\_ClockRoot\_MuxOscRc24M* ASRC mux from OscRc24M.  
*kCLOCK\_ASRC\_ClockRoot\_MuxOscRc400M* ASRC mux from OscRc400M.  
*kCLOCK\_ASRC\_ClockRoot\_MuxSysPll3Out* ASRC mux from SysPll3Out.  
*kCLOCK\_ASRC\_ClockRoot\_MuxAudioPllOut* ASRC mux from AudioPllOut.  
*kCLOCK\_MIC\_ClockRoot\_MuxOscRc24M* MIC mux from OscRc24M.  
*kCLOCK\_MIC\_ClockRoot\_MuxOscRc400M* MIC mux from OscRc400M.  
*kCLOCK\_MIC\_ClockRoot\_MuxSysPll3Div2* MIC mux from SysPll3Div2.  
*kCLOCK\_MIC\_ClockRoot\_MuxAudioPllOut* MIC mux from AudioPllOut.  
*kCLOCK\_CKO1\_ClockRoot\_MuxOscRc24M* CKO1 mux from OscRc24M.  
*kCLOCK\_CKO1\_ClockRoot\_MuxOscRc400M* CKO1 mux from OscRc400M.  
*kCLOCK\_CKO1\_ClockRoot\_MuxSysPll3Div2* CKO1 mux from SysPll3Div2.  
*kCLOCK\_CKO1\_ClockRoot\_MuxSysPll1Div2* CKO1 mux from SysPll1Div2.  
*kCLOCK\_CKO2\_ClockRoot\_MuxOscRc24M* CKO2 mux from OscRc24M.  
*kCLOCK\_CKO2\_ClockRoot\_MuxOscRc400M* CKO2 mux from OscRc400M.  
*kCLOCK\_CKO2\_ClockRoot\_MuxSysPll1Div5* CKO2 mux from SysPll1Div5.  
*kCLOCK\_CKO2\_ClockRoot\_MuxArmPllOut* CKO2 mux from ArmPllOut.

#### 4.4.5 enum clock\_osc\_t

Enumerator

*kCLOCK\_RcOsc* On chip OSC.  
*kCLOCK\_XtalOsc* 24M Xtal OSC

#### 4.4.6 enum clock\_gate\_value\_t

Enumerator

*kCLOCK\_Off* Clock is off.

*kCLOCK\_On* Clock is on.

#### 4.4.7 enum clock\_mode\_t

Enumerator

*kCLOCK\_ModeRun* Remain in run mode.

*kCLOCK\_ModeWait* Transfer to wait mode.

*kCLOCK\_ModeStop* Transfer to stop mode.

#### 4.4.8 enum clock\_usb\_src\_t

Enumerator

*kCLOCK\_Usb480M* Use 480M.

*kCLOCK\_UsbSrcUnused* Used when the function does not care the clock source.

#### 4.4.9 enum clock\_usb\_phy\_src\_t

Enumerator

*kCLOCK\_Usbphy480M* Use 480M.

#### 4.4.10 enum \_clock\_pll\_clk\_src

Enumerator

*kCLOCK\_PllClkSrc24M* Pll clock source 24M.

*kCLOCK\_PllSrcClkPN* Pll clock source CLK1\_P and CLK1\_N.

#### 4.4.11 enum clock\_pll\_post\_div\_t

Enumerator

*kCLOCK\_PllPostDiv2* Divide by 2.

- kCLOCK\_PlPostDiv4* Divide by 4.
- kCLOCK\_PlPostDiv8* Divide by 8.
- kCLOCK\_PlPostDiv1* Divide by 1.

#### 4.4.12 enum clock\_pll\_t

Enumerator

- kCLOCK\_PlArm* ARM PLL.
- kCLOCK\_PlSys1* SYS1 PLL, it has a dedicated frequency of 1GHz.
- kCLOCK\_PlSys2* SYS2 PLL, it has a dedicated frequency of 528MHz.
- kCLOCK\_PlSys3* SYS3 PLL, it has a dedicated frequency of 480MHz.
- kCLOCK\_PlAudio* Audio PLL.
- kCLOCK\_PlInvalid* Invalid value.

#### 4.4.13 enum clock\_pfd\_t

Enumerator

- kCLOCK\_Pfd0* PLL PFD0.
- kCLOCK\_Pfd1* PLL PFD1.
- kCLOCK\_Pfd2* PLL PFD2.
- kCLOCK\_Pfd3* PLL PFD3.

#### 4.4.14 enum clock\_control\_mode\_t

Enumerator

- kCLOCK\_SoftwareMode* Software control mode.
- kCLOCK\_GpcMode* GPC control mode.

#### 4.4.15 enum clock\_24MOsc\_mode\_t

Enumerator

- kCLOCK\_24MOscHighGainMode* 24MHz crystal oscillator work as high gain mode.
- kCLOCK\_24MOscBypassMode* 24MHz crystal oscillator work as bypass mode.
- kCLOCK\_24MOscLowPowerMode* 24MHz crystal oscillator work as low power mode.

#### 4.4.16 enum clock\_1MHzOut\_behavior\_t

Enumerator

***kCLOCK\_1MHzOutDisable*** Disable 1MHz output clock.

***kCLOCK\_1MHzOutEnableLocked1Mhz*** Enable 1MHz output clock, and select locked 1MHz to output.

***kCLOCK\_1MHzOutEnableFreeRunning1Mhz*** Enable 1MHz output clock, and select free-running 1MHz to output.

#### 4.4.17 enum clock\_level\_t

Enumerator

***kCLOCK\_Level0*** Not needed in any mode.

***kCLOCK\_Level1*** Needed in RUN mode.

***kCLOCK\_Level2*** Needed in RUN and WAIT mode.

***kCLOCK\_Level3*** Needed in RUN, WAIT and STOP mode.

***kCLOCK\_Level4*** Always on in any mode.

### 4.5 Function Documentation

#### 4.5.1 static void CLOCK\_SetRootClockMux ( *clock\_root\_t root*, *uint8\_t src* ) [inline], [static]

Parameters

|             |                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------|
| <i>root</i> | Which root clock node to set, see <a href="#">clock_root_t</a> .                                               |
| <i>src</i>  | Clock mux value to set, different mux has different value range. See <a href="#">clock_root_mux_source_t</a> . |

#### 4.5.2 static uint32\_t CLOCK\_GetRootClockMux ( *clock\_root\_t root* ) [inline], [static]

Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>root</i> | Which root clock node to get, see <a href="#">clock_root_t</a> . |
|-------------|------------------------------------------------------------------|

Returns

Clock mux value.

#### 4.5.3 static **clock\_name\_t CLOCK\_GetRootClockSource ( *clock\_root\_t root, uint32\_t src* ) [inline], [static]**

Parameters

|             |                                                                       |
|-------------|-----------------------------------------------------------------------|
| <i>root</i> | Which root clock node to get, see <a href="#">clock_root_t</a> .      |
| <i>src</i>  | Clock mux value to get, see <a href="#">clock_root_mux_source_t</a> . |

Returns

Clock source

#### 4.5.4 static void **CLOCK\_SetRootClockDiv ( *clock\_root\_t root, uint32\_t div* ) [inline], [static]**

Parameters

|             |                                                                      |
|-------------|----------------------------------------------------------------------|
| <i>root</i> | Which root clock to set, see <a href="#">clock_root_t</a> .          |
| <i>div</i>  | Clock div value to set, different divider has different value range. |

#### 4.5.5 static **uint32\_t CLOCK\_GetRootClockDiv ( *clock\_root\_t root* ) [inline], [static]**

Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>root</i> | Which root clock node to get, see <a href="#">clock_root_t</a> . |
|-------------|------------------------------------------------------------------|

Returns

divider set for this root

4.5.6 **static void CLOCK\_PowerOffRootClock ( clock\_root\_t *root* ) [inline], [static]**

Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>root</i> | Which root clock node to set, see <a href="#">clock_root_t</a> . |
|-------------|------------------------------------------------------------------|

#### 4.5.7 static void CLOCK\_PowerOnRootClock ( *clock\_root\_t root* ) [inline], [static]

Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>root</i> | Which root clock node to set, see <a href="#">clock_root_t</a> . |
|-------------|------------------------------------------------------------------|

#### 4.5.8 static void CLOCK\_SetRootClock ( *clock\_root\_t root*, const *clock\_root\_config\_t \* config* ) [inline], [static]

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>root</i>   | Which root clock node to set, see <a href="#">clock_root_t</a> . |
| <i>config</i> | root clock config, see <a href="#">clock_root_config_t</a>       |

#### 4.5.9 static void CLOCK\_ControlGate ( *clock\_ip\_name\_t name*, *clock\_gate\_value\_t value* ) [inline], [static]

Note

This API will not have any effect when this clock is in CPULPM or SetPoint Mode

Parameters

|              |                                                                   |
|--------------|-------------------------------------------------------------------|
| <i>name</i>  | Which clock to enable, see <a href="#">clock_lpcg_t</a> .         |
| <i>value</i> | Clock gate value to set, see <a href="#">clock_gate_value_t</a> . |

#### 4.5.10 static void CLOCK\_EnableClock ( *clock\_ip\_name\_t name* ) [inline], [static]

Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>name</i> | Which clock to enable, see <a href="#">clock_lpcg_t</a> . |
|-------------|-----------------------------------------------------------|

#### 4.5.11 static void CLOCK\_DisableClock ( *clock\_ip\_name\_t name* ) [inline], [static]

Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>name</i> | Which clock to disable, see <a href="#">clock_lpcg_t</a> . |
|-------------|------------------------------------------------------------|

#### 4.5.12 uint32\_t CLOCK\_GetFreq ( *clock\_name\_t name* )

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

Parameters

|             |                                            |
|-------------|--------------------------------------------|
| <i>name</i> | Clock names defined in <i>clock_name_t</i> |
|-------------|--------------------------------------------|

Returns

Clock frequency value in hertz

#### 4.5.13 static uint32\_t CLOCK\_GetRootClockFreq ( *clock\_root\_t root* ) [inline], [static]

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

Parameters

|             |                                            |
|-------------|--------------------------------------------|
| <i>root</i> | Clock names defined in <i>clock_root_t</i> |
|-------------|--------------------------------------------|

Returns

Clock frequency value in hertz

**4.5.14 uint32\_t CLOCK\_GetM7Freq( void )**

Returns

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

**4.5.15 uint32\_t CLOCK\_GetM33Freq( void )**

Returns

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

**4.5.16 static bool CLOCK\_IsPllBypassed( clock\_pll\_t *pll* ) [inline], [static]**

Parameters

|            |                                                                |
|------------|----------------------------------------------------------------|
| <i>pll</i> | PLL control name (see <a href="#">clock_pll_t</a> enumeration) |
|------------|----------------------------------------------------------------|

Returns

PLL bypass status.

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

**4.5.17 static bool CLOCK\_IsPllEnabled( clock\_pll\_t *pll* ) [inline], [static]**

Parameters

|            |                                                                |
|------------|----------------------------------------------------------------|
| <i>pll</i> | PLL control name (see <a href="#">clock_pll_t</a> enumeration) |
|------------|----------------------------------------------------------------|

Returns

PLL bypass status.

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

**4.5.18 static uint32\_t CLOCK\_GetRtcFreq( void ) [inline], [static]**

Returns

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

**4.5.19 void CLOCK\_SetClockSourceControlMode( clock\_name\_t name, clock\_control\_mode\_t controlMode )**

Parameters

|                    |                                                                                    |
|--------------------|------------------------------------------------------------------------------------|
| <i>name</i>        | Clock names defined in <code>clock_name_t</code>                                   |
| <i>controlMode</i> | The control mode to be set, please refer to <a href="#">clock_control_mode_t</a> . |

**4.5.20 static void CLOCK\_OSC\_EnableOscRc24M( bool enable ) [inline], [static]**

Parameters

|               |                                                                                                                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enable</i> | Used to enable or disable the 24MHz RC oscillator. <ul style="list-style-type: none"> <li>• <b>true</b> Enable the 24MHz RC oscillator.</li> <li>• <b>false</b> Disable the 24MHz RC oscillator.</li> </ul> |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**4.5.21 void CLOCK\_OSC\_EnableOsc24M( void )**

This function enables OSC 24Mhz.

**4.5.22 static void CLOCK\_OSC\_GateOsc24M( bool enableGate ) [inline], [static]**

Note

Gating the 24MHz crystal oscillator can save power.

Parameters

|                   |                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enableGate</i> | Used to gate/ungate the 24MHz crystal oscillator. <ul style="list-style-type: none"> <li>• <b>true</b> Gate the 24MHz crystal oscillator to save power.</li> <li>• <b>false</b> Ungate the 24MHz crystal oscillator.</li> </ul> |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 4.5.23 void CLOCK\_OSC\_SetOsc24MWorkMode ( `clock_24MOsc_mode_t workMode` )

Parameters

|                 |                                                                                                             |
|-----------------|-------------------------------------------------------------------------------------------------------------|
| <i>workMode</i> | The work mode of 24MHz crystal oscillator, please refer to <a href="#">clock_24MOsc_mode_t</a> for details. |
|-----------------|-------------------------------------------------------------------------------------------------------------|

#### 4.5.24 void CLOCK\_OSC\_EnableOscRc400M ( `void` )

This function enables OSC RC 400Mhz.

#### 4.5.25 static void CLOCK\_OSC\_GateOscRc400M ( `bool enableGate` ) [inline], [static]

Parameters

|                   |                                                                                                                                                                                                   |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enableGate</i> | Used to gate/ungate 400MHz RC oscillator. <ul style="list-style-type: none"> <li>• <b>true</b> Gate the 400MHz RC oscillator.</li> <li>• <b>false</b> Ungate the 400MHz RC oscillator.</li> </ul> |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 4.5.26 void CLOCK\_OSC\_TrimOscRc400M ( `bool enable, bool bypass, uint16_t trim` )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>enable</i> | Used to enable trim function. |
| <i>bypass</i> | Bypass the trim function.     |
| <i>trim</i>   | Trim value.                   |

#### 4.5.27 void CLOCK\_OSC\_SetOscRc400MRefClkDiv ( uint8\_t *divValue* )

Note

slow\_clk = ref\_clk / (divValue + 1), and the recommand divide value is 24.

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>divValue</i> | The divide value to be set, the available range is 0~63. |
|-----------------|----------------------------------------------------------|

#### 4.5.28 void CLOCK\_OSC\_SetOscRc400MFastClkCount ( uint16\_t *targetCount* )

Parameters

|                    |                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------|
| <i>targetCount</i> | The desired target for the fast clock, should be the number of clock cycles of the fast_clk per divided ref_clk. |
|--------------------|------------------------------------------------------------------------------------------------------------------|

#### 4.5.29 void CLOCK\_OSC\_SetOscRc400MHysteresisValue ( uint8\_t *negHysteresis*, uint8\_t *posHysteresis* )

Note

The hysteresis value should be set after the clock is tuned.

Parameters

|                      |                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------|
| <i>negHysteresis</i> | The negative hysteresis value for the turned clock, this value in number of clock cycles of the fast clock |
| <i>posHysteresis</i> | The positive hysteresis value for the turned clock, this value in number of clock cycles of the fast clock |

#### 4.5.30 void CLOCK\_OSC\_BypassOscRc400MTuneLogic ( bool *enableBypass* )

Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enableBypass</i> | Used to control whether to bypass the turn logic. <ul style="list-style-type: none"> <li>• <b>true</b> Bypass the tune logic and use the programmed oscillator frequency to run the oscillator. Function <a href="#">CLOCK_OSC_SetOscRc400MTuneValue()</a> can be used to set oscillator frequency.</li> <li>• <b>false</b> Use the output of tune logic to run the oscillator.</li> </ul> |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**4.5.31 void CLOCK\_OSC\_EnableOscRc400MTuneLogic ( bool *enable* )**

Parameters

|               |                                                                                                                                                                                    |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enable</i> | Used to start or stop the tune logic. <ul style="list-style-type: none"> <li>• <b>true</b> Start tuning</li> <li>• <b>false</b> Stop tuning and reset the tuning logic.</li> </ul> |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**4.5.32 void CLOCK\_OSC\_FreezeOscRc400MTuneValue ( bool *enableFreeze* )**

Parameters

|                     |                                                                                                                                                                                                                                                                                    |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enableFreeze</i> | Used to control whether to freeze the tune value. <ul style="list-style-type: none"> <li>• <b>true</b> Freeze the tune at the current tuned value and the oscillator runs at the frozen tune value.</li> <li>• <b>false</b> Unfreezes and continues the tune operation.</li> </ul> |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**4.5.33 void CLOCK\_OSC\_SetOscRc400MTuneValue ( uint8\_t *tuneValue* )**

Parameters

|                  |                                                          |
|------------------|----------------------------------------------------------|
| <i>tuneValue</i> | The tune value to determine the frequency of Oscillator. |
|------------------|----------------------------------------------------------|

**4.5.34 void CLOCK\_OSC\_Set1MHzOutputBehavior ( clock\_1MHzOut\_behavior\_t *behavior* )**

## Note

The 1MHz clock is divided from 400M RC Oscillator.

## Parameters

|                 |                                                                                                          |
|-----------------|----------------------------------------------------------------------------------------------------------|
| <i>behavior</i> | The behavior of 1MHz output clock, please refer to <a href="#">clock_1MHzOut_behavior_t</a> for details. |
|-----------------|----------------------------------------------------------------------------------------------------------|

**4.5.35 void CLOCK\_OSC\_SetLocked1MHzCount ( uint16\_t *count* )**

## Parameters

|              |                                                                                                                                          |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>count</i> | Used to set the desired target for the locked 1MHz clock out, the value in number of clock cycles of the fast clock per divided ref_clk. |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------|

**4.5.36 bool CLOCK\_OSC\_CheckLocked1MHzErrorFlag ( void )**

## Returns

The error flag for locked 1MHz clock out.

- **true** The count value has been reached within one diviced ref clock period
- **false** No effect.

**4.5.37 uint16\_t CLOCK\_OSC\_GetCurrentOscRc400MFastClockCount ( void )**

## Returns

The current count for the fast clock.

**4.5.38 uint8\_t CLOCK\_OSC\_GetCurrentOscRc400MTuneValue ( void )**

## Returns

The current tune value.

**4.5.39 void CLOCK\_InitArmPII ( const clock\_arm\_pll\_config\_t \* *config* )**

This function initialize the ARM PLL with specific settings

Parameters

|               |                              |
|---------------|------------------------------|
| <i>config</i> | configuration to set to PLL. |
|---------------|------------------------------|

#### 4.5.40 status\_t CLOCK\_CalcArmPIIFreq ( `clock_arm_pll_config_t * config,` `uint32_t freqInMhz` )

This function calculates config values per given frequency for Arm PLL

Parameters

|                  |                      |
|------------------|----------------------|
| <i>config</i>    | pll config structure |
| <i>freqInMhz</i> | target frequency     |

#### 4.5.41 status\_t CLOCK\_InitArmPIIWithFreq ( `uint32_t freqInMhz` )

This function initializes the Arm PLL with specific frequency

Parameters

|                  |                  |
|------------------|------------------|
| <i>freqInMhz</i> | target frequency |
|------------------|------------------|

#### 4.5.42 void CLOCK\_CalcPIISpreadSpectrum ( `uint32_t factor,` `uint32_t range,` `uint32_t mod,` `clock_pll_ss_config_t * ss` )

This function calculate spread spectrum step and stop according to given parameters. For integer PLL (syspll2) the factor is mfd, while for other fractional PLLs (audio/syspll1), the factor is denominator.

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>factor</i> | factor to calculate step/stop        |
| <i>range</i>  | spread spectrum range                |
| <i>mod</i>    | spread spectrum modulation frequency |
| <i>ss</i>     | calculated spread spectrum values    |

#### 4.5.43 void CLOCK\_InitSysPLL1 ( `const clock_sys_pll1_config_t * config` )

This function initializes the System PLL1 with specific settings

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>config</i> | Configuration to set to PLL1. |
|---------------|-------------------------------|

#### 4.5.44 void CLOCK\_InitSysPll2 ( const clock\_sys\_pll2\_config\_t \* *config* )

This function initializes the System PLL2 with specific settings

Parameters

|               |                                                                                                              |
|---------------|--------------------------------------------------------------------------------------------------------------|
| <i>config</i> | Configuration to configure spread spectrum. This parameter can be NULL, if no need to enable spread spectrum |
|---------------|--------------------------------------------------------------------------------------------------------------|

#### 4.5.45 bool CLOCK\_IsSysPll2PfdEnabled ( clock\_pfd\_t *pfd* )

Parameters

|            |                  |
|------------|------------------|
| <i>pfd</i> | PFD control name |
|------------|------------------|

Returns

PFD bypass status.

- true: power on.
- false: power off.

Note

Only useful in software control mode.

#### 4.5.46 void CLOCK\_InitSysPll3 ( void )

This function initializes the System PLL3 with specific settings

#### 4.5.47 bool CLOCK\_IsSysPll3PfdEnabled ( clock\_pfd\_t *pfd* )

Parameters

|            |                  |
|------------|------------------|
| <i>pfd</i> | PFD control name |
|------------|------------------|

Returns

PFD bypass status.

- true: power on.
- false: power off.

Note

Only useful in software control mode.

#### 4.5.48 void CLOCK\_SetPllBypass ( *clock\_pll\_t pll*, *bool bypass* )

Parameters

|               |                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>pll</i>    | PLL control name (see <a href="#">clock_pll_t</a> enumeration)                                                                  |
| <i>bypass</i> | Bypass the PLL. <ul style="list-style-type: none"> <li>• true: Bypass the PLL.</li> <li>• false: Not bypass the PLL.</li> </ul> |

#### 4.5.49 status\_t CLOCK\_CalcAudioPllFreq ( *clock\_audio\_pll\_config\_t \* config*, *uint32\_t freqInMhz* )

This function calculates config values per given frequency for Audio PLL.

Parameters

|                  |                      |
|------------------|----------------------|
| <i>config</i>    | pll config structure |
| <i>freqInMhz</i> | target frequency     |

#### 4.5.50 status\_t CLOCK\_InitAudioPllWithFreq ( *uint32\_t freqInMhz*, *bool ssEnable*, *uint32\_t ssRange*, *uint32\_t ssMod* )

This function initializes the Audio PLL with specific frequency

Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>freqInMhz</i> | target frequency                     |
| <i>ssEnable</i>  | enable spread spectrum or not        |
| <i>ssRange</i>   | range spread spectrum range          |
| <i>ssMod</i>     | spread spectrum modulation frequency |

#### 4.5.51 void CLOCK\_InitAudioPII ( const clock\_audio\_pll\_config\_t \* *config* )

This function initializes the Audio PLL with specific settings

Parameters

|               |                              |
|---------------|------------------------------|
| <i>config</i> | Configuration to set to PLL. |
|---------------|------------------------------|

#### 4.5.52 uint32\_t CLOCK\_GetPIIFreq ( clock\_pll\_t *pll* )

This function get current output frequency of specific PLL

Parameters

|            |                            |
|------------|----------------------------|
| <i>pll</i> | pll name to get frequency. |
|------------|----------------------------|

Returns

The PLL output frequency in hertz.

#### 4.5.53 void CLOCK\_InitPfd ( clock\_pll\_t *pll*, clock\_pfd\_t *pfd*, uint8\_t *frac* )

This function initializes the System PLL PFD. During new value setting, the clock output is disabled to prevent glitch.

Parameters

|            |                                            |
|------------|--------------------------------------------|
| <i>pll</i> | Which PLL of targeting PFD to be operated. |
|------------|--------------------------------------------|

|             |                            |
|-------------|----------------------------|
| <i>pfld</i> | Which PFD clock to enable. |
| <i>frac</i> | The PFD FRAC value.        |

## Note

It is recommended that PFD settings are kept between 13-35.

**4.5.54 void CLOCK\_DeinitPfd ( *clock\_pll\_t pll*, *clock\_pfd\_t pfd* )**

## Parameters

|            |                                            |
|------------|--------------------------------------------|
| <i>pll</i> | Which PLL of targeting PFD to be operated. |
| <i>pfd</i> | Which PFD clock to enable.                 |

**4.5.55 uint32\_t CLOCK\_GetPfdFreq ( *clock\_pll\_t pll*, *clock\_pfd\_t pfd* )**

This function get current output frequency of specific System PLL PFD

## Parameters

|            |                                            |
|------------|--------------------------------------------|
| <i>pll</i> | Which PLL of targeting PFD to be operated. |
| <i>pfd</i> | pfd name to get frequency.                 |

## Returns

The PFD output frequency in hertz.

**4.5.56 bool CLOCK\_EnableUsbhs0Clock ( *clock\_usb\_src\_t src*, *uint32\_t freq* )**

This function only enables the access to USB HS prepheral, upper layer should first call the [CLOCK\\_EnableUsbhs0PhyPllClock](#) to enable the PHY clock to use USB HS.

## Parameters

|             |                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------|
| <i>src</i>  | USB HS does not care about the clock source, here must be <a href="#">kCLOCK_UsbSrcUnused</a> . |
| <i>freq</i> | USB HS does not care about the clock source, so this parameter is ignored.                      |

Return values

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>true</i>  | The clock is set successfully.                          |
| <i>false</i> | The clock source is invalid to get proper USB HS clock. |

#### 4.5.57 bool CLOCK\_EnableUsbhs1Clock ( **clock\_usb\_src\_t src, uint32\_t freq** )

This function only enables the access to USB HS prepheral, upper layer should first call the [CLOCK\\_EnableUsbhs0PhyPllClock](#) to enable the PHY clock to use USB HS.

Parameters

|             |                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------|
| <i>src</i>  | USB HS does not care about the clock source, here must be <a href="#">kCLOCK_UsbSrcUnused</a> . |
| <i>freq</i> | USB HS does not care about the clock source, so this parameter is ignored.                      |

Return values

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>true</i>  | The clock is set successfully.                          |
| <i>false</i> | The clock source is invalid to get proper USB HS clock. |

#### 4.5.58 bool CLOCK\_EnableUsbhs0PhyPllClock ( **clock\_usb\_phy\_src\_t src, uint32\_t freq** )

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>src</i>  | USB HS PHY PLL clock source.    |
| <i>freq</i> | The frequency specified by src. |

Return values

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>true</i>  | The clock is set successfully.                          |
| <i>false</i> | The clock source is invalid to get proper USB HS clock. |

#### 4.5.59 void CLOCK\_DisableUsbhs0PhyPllClock ( void )

This function disables USB HS PHY PLL clock.

#### 4.5.60 bool CLOCK\_EnableUsbhs1PhyPllClock ( *clock\_usb\_phy\_src\_t src*, *uint32\_t freq* )

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>src</i>  | USB HS PHY PLL clock source.    |
| <i>freq</i> | The frequency specified by src. |

Return values

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>true</i>  | The clock is set successfully.                          |
| <i>false</i> | The clock source is invalid to get proper USB HS clock. |

#### 4.5.61 void CLOCK\_DisableUsbhs1PhyPllClock ( void )

This function disables USB HS PHY PLL clock.

#### 4.5.62 static void CLOCK\_OSCPLL\_LockWhiteList ( *clock\_name\_t name* ) [*inline*], [*static*]

Note

Once locked, this bit and domain ID white list can not be changed until next system reset.

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>name</i> | Clock source name, see <a href="#">clock_name_t</a> . |
|-------------|-------------------------------------------------------|

#### 4.5.63 static void CLOCK\_OSCPLL\_SetWhiteList ( *clock\_name\_t name*, *uint8\_t domainId* ) [inline], [static]

Note

If LOCK\_LIST bit is set, domain ID white list can not be changed until next system reset.

Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>name</i>     | Clock source name, see <a href="#">clock_name_t</a> . |
| <i>domainId</i> | Domains that on the whitelist can change this clock.  |

#### 4.5.64 void CLOCK\_OSCPLL\_ControlByCpuLowPowerMode ( *clock\_name\_t name*, *uint32\_t domainMap*, *clock\_level\_t level* )

Note

When LOCK\_MODE bit is set, control mode can not be changed until next system reset.

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>name</i>      | Clock source name, see <a href="#">clock_name_t</a> . |
| <i>domainMap</i> | Domains that on the whitelist can change this clock.  |
| <i>level</i>     | Depend level of this clock.                           |

#### 4.5.65 static void CLOCK\_ROOT\_LockWhiteList ( *clock\_root\_t name* ) [inline], [static]

Note

Once locked, this bit and domain ID white list can not be changed until next system reset.

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>name</i> | Clock root name, see <a href="#">clock_root_t</a> . |
|-------------|-----------------------------------------------------|

#### 4.5.66 static void CLOCK\_ROOT\_SetWhiteList ( *clock\_root\_t name*, *uint8\_t domainId* ) [inline], [static]

Note

If LOCK\_LIST bit is set, domain ID white list can not be changed until next system reset.

Parameters

|                 |                                                      |
|-----------------|------------------------------------------------------|
| <i>name</i>     | Clock root name, see <a href="#">clock_root_t</a> .  |
| <i>domainId</i> | Domains that on the whitelist can change this clock. |

#### 4.5.67 static void CLOCK\_LPCG\_LockWhiteList ( *clock\_lpcg\_t name* ) [inline], [static]

Note

Once locked, this bit and domain ID white list can not be changed until next system reset.

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>name</i> | Clock gate name, see <a href="#">clock_lpcg_t</a> . |
|-------------|-----------------------------------------------------|

#### 4.5.68 static void CLOCK\_LPCG\_SetWhiteList ( *clock\_lpcg\_t name*, *uint8\_t domainId* ) [inline], [static]

Note

If LOCK\_LIST bit is set, domain ID white list can not be changed until next system reset.

Parameters

|                 |                                                      |
|-----------------|------------------------------------------------------|
| <i>name</i>     | Clock gate name, see <a href="#">clock_lpcg_t</a> .  |
| <i>domainId</i> | Domains that on the whitelist can change this clock. |

**4.5.69 void CLOCK\_LPCG\_ControlByCpuLowPowerMode ( *clock\_lpcg\_t name*,  
*uint32\_t domainMap*, *clock\_level\_t level* )**

Note

When LOCK\_MODE bit is set, control mode can not be changed until next system reset.

Parameters

|                  |                                                      |
|------------------|------------------------------------------------------|
| <i>name</i>      | Clock gate name, see <a href="#">clock_lpcg_t</a> .  |
| <i>domainMap</i> | Domains that on the whitelist can change this clock. |
| <i>level</i>     | Depend level of this clock.                          |

# Chapter 5

## DCDC\_SOC driver

### 5.1 Overview

The MCUXpresso SDK provides APIs for MCUXpresso SDK devices' dc当地操作。

The DCDC Converter(DCDC) is used for generating the power supply for the chip's core logic. It can provide two outputs for SoC peripherals and external devices with high conversion efficiency. As a module to provide the power for hardware system, the DCDC starts working when the system is powered up before the software takes over the SoC. Some important configuration is done by the board settings. Before the software can access the DCDC's register, the DCDC is already working normally with the default settings. However, if the application needs to improve the DCDC's performance or change the default settings, this DCDC driver would be helpful. The DCDC's register cannot be accessed by software before its initialization (open the clock gate).

The DCDC module supports two control modes: Static control mode and Set point control mode. When the module controlled by GPC set points, that is set point control mode, the DCDC\_SetPointInit() function can be used to enable DCDC module, set outputs' target voltage in the specific set point. When the control mode is set as static control mode, to change the target voltage value of DCDC's outputs, the APIs in "Outputs' Target Voltage Related Interfaces" function group can be used. To set DCDC module's power mode, the APIs in "Power Mode Related Interfaces" function group can be used.

### Data Structures

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

### Macros

- #define `FSL_DCDC_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*DCDC driver version.*
- #define `VDD1P0_TARGET_VOLTAGE`  
*The array of VDD1P0 target voltage.*
- #define `VDD1P8_TARGET_VOLTAGE`  
*The array of VDD1P8 target voltage.*

## Enumerations

- enum `_dcdc_status_flags` { `kDCDC_AlreadySettledStatusFlag` = DCDC\_REG0\_STS\_DC\_OK\_M-ASK }
- The enumeration of DCDC status flags.*
- enum `dcdc_core_slice_t` {  
    `kDCDC_CORE0` = 0x0U,  
    `kDCDC_CORE1` = 0x1U }
- CORE slice.*
- enum `dcdc_control_mode_t` {  
    `kDCDC_SoftwareControl` = 0U,  
    `kDCDC_GPCControl` = 1U }
- DCDC control mode, including software control mode and GPC control mode.*
- enum `dcdc_trim_input_mode_t` {  
    `kDCDC_SampleTrimInput` = 0U,  
    `kDCDC_HoldTrimInput` = 1U }
- DCDC trim input mode, including sample trim input and hold trim input.*
- enum `dcdc_1P0_target_vol_t` {

```
kDCDC_1P0Target0P6V = 0U,
kDCDC_1P0Target0P625V,
kDCDC_1P0Target0P65V,
kDCDC_1P0Target0P675V,
kDCDC_1P0Target0P7V,
kDCDC_1P0Target0P725V,
kDCDC_1P0Target0P75V,
kDCDC_1P0Target0P775V,
kDCDC_1P0Target0P8V,
kDCDC_1P0Target0P825V,
kDCDC_1P0Target0P85V,
kDCDC_1P0Target0P875V,
kDCDC_1P0Target0P9V,
kDCDC_1P0Target0P925V,
kDCDC_1P0Target0P95V,
kDCDC_1P0Target0P975V,
kDCDC_1P0Target1P0V,
kDCDC_1P0Target1P025V,
kDCDC_1P0Target1P05V,
kDCDC_1P0Target1P075V,
kDCDC_1P0Target1P1V,
kDCDC_1P0Target1P125V,
kDCDC_1P0Target1P15V,
kDCDC_1P0Target1P175V,
kDCDC_1P0Target1P2V,
kDCDC_1P0Target1P225V,
kDCDC_1P0Target1P25V,
kDCDC_1P0Target1P275V,
kDCDC_1P0Target1P3V,
kDCDC_1P0Target1P325V,
kDCDC_1P0Target1P35V,
kDCDC_1P0Target1P375V = 0x1FU }
```

*The enumeration VDD1P0's target voltage value.*

- enum `dcdc_1P8_target_vol_t` {

```
kDCDC_1P8Target1P5V = 0U,
kDCDC_1P8Target1P525V,
kDCDC_1P8Target1P55V,
kDCDC_1P8Target1P575V,
kDCDC_1P8Target1P6V,
kDCDC_1P8Target1P625V,
kDCDC_1P8Target1P65V,
kDCDC_1P8Target1P675V,
kDCDC_1P8Target1P7V,
kDCDC_1P8Target1P725V,
kDCDC_1P8Target1P75V,
kDCDC_1P8Target1P775V,
kDCDC_1P8Target1P8V,
kDCDC_1P8Target1P825V,
kDCDC_1P8Target1P85V,
kDCDC_1P8Target1P875V,
kDCDC_1P8Target1P9V,
kDCDC_1P8Target1P925V,
kDCDC_1P8Target1P95V,
kDCDC_1P8Target1P975V,
kDCDC_1P8Target2P0V,
kDCDC_1P8Target2P025V,
kDCDC_1P8Target2P05V,
kDCDC_1P8Target2P075V,
kDCDC_1P8Target2P1V,
kDCDC_1P8Target2P125V,
kDCDC_1P8Target2P15V,
kDCDC_1P8Target2P175V,
kDCDC_1P8Target2P2V,
kDCDC_1P8Target2P225V,
kDCDC_1P8Target2P25V,
kDCDC_1P8Target2P275V = 0x1FU }
```

*The enumeration VDD1P8's target voltage value.*

- enum `dcdc_comparator_current_bias_t` {
 

```
kDCDC_ComparatorCurrentBias50nA = 0U,
kDCDC_ComparatorCurrentBias100nA = 1U,
kDCDC_ComparatorCurrentBias200nA = 2U,
kDCDC_ComparatorCurrentBias400nA = 3U }
```

*The current bias of low power comparator.*

- enum `dcdc_peak_current_threshold_t` {
 

```
kDCDC_PeakCurrentRunMode250mALPMode1P5A = 0U,
kDCDC_PeakCurrentRunMode200mALPMode1P5A,
kDCDC_PeakCurrentRunMode250mALPMode2A,
kDCDC_PeakCurrentRunMode200mALPMode2A }
```

*The threshold if peak current detection.*

- enum `dcdc_clock_source_t` {
   
    `kDCDC_ClockAutoSwitch` = 0U,  
`kDCDC_ClockInternalOsc` = 1U,  
`kDCDC_ClockExternalOsc` = 2U }

*Oscillator clock option.*

- enum `dcdc_voltage_output_sel_t` {
   
    `kDCDC_VoltageOutput1P8` = 0U,  
`kDCDC_VoltageOutput1P0` = 1U }

*Voltage output option.*

## Initialization and De-initialization Interfaces

- void `DCDC_Init` (DCDC\_Type \*base, const `dcdc_config_t` \*config)  
*Initializes the basic resource of DCDC module, such as control mode, etc.*
- void `DCDC_Deinit` (DCDC\_Type \*base)  
*De-initializes the DCDC module.*
- void `DCDC_GetDefaultConfig` (`dcdc_config_t` \*config)  
*Gets the default setting for DCDC, such as control mode, etc.*

## Outputs' Target Voltage Related Interfaces

- static void `DCDC_SetVDD1P0LowPowerModeTargetVoltage` (DCDC\_Type \*base, `dcdc_core_slice_t` core, `dcdc_1P0_target_vol_t` targetVoltage)
- static uint16\_t `DCDC_GetVDD1P0LowPowerModeTargetVoltage` (DCDC\_Type \*base)  
*Gets the target value of VDD1P0 in low power mode, the result takes "mV" as the unit.*
- static void `DCDC_EnableVDD1P0LowPowerMode` (DCDC\_Type \*base, `dcdc_core_slice_t` core, bool enable)
- static void `DCDC_SetVDD1P0BuckModeTargetVoltage` (DCDC\_Type \*base, `dcdc_core_slice_t` core, `dcdc_1P0_target_vol_t` targetVoltage)  
*Sets the target value(ranges from 0.6V to 1.375V) of VDD1P0 in buck mode, 25mV each step.*
- static uint16\_t `DCDC_GetVDD1P0BuckModeTargetVoltage` (DCDC\_Type \*base)  
*Gets the target value of VDD1P0 in buck mode, the result takes "mV" as the unit.*
- static void `DCDC_GPC_SetVDD1P0BuckModeTargetVoltage` (DCDC\_Type \*base, `dcdc_core_slice_t` core, `dcdc_1P0_target_vol_t` targetVoltage)
- static void `DCDC_GPC_SetVDD1P0LowPowerModeTargetVoltage` (DCDC\_Type \*base, `dcdc_core_slice_t` core, `dcdc_1P0_target_vol_t` targetVoltage)  
*Sets the target value(ranges from 0.625V to 1.4V) of VDD1P0 in low power mode, 25mV each step.*
- static void `DCDC_GPC_EnableVDD1P0LowPowerMode` (DCDC\_Type \*base, `dcdc_core_slice_t` core, bool enable)  
*Enable VDD1P0 in low power mode.*
- static void `DCDC_SetVDD1P8TargetVoltage` (DCDC\_Type \*base, `dcdc_core_slice_t` core, `dcdc_1P8_target_vol_t` targetVoltage)  
*Sets the target value(ranges from 1.5V to 2.275V) of VDD1P8, 25mV each step.*
- static uint16\_t `DCDC_GetVDD1P8TargetVoltage` (DCDC\_Type \*base)  
*Gets the target value of VDD1P8, the result takes "mV" as the unit.*
- static void `DCDC_EnableVDD1P0TargetVoltageStepping` (DCDC\_Type \*base, bool enable)  
*Enables/Disables stepping for VDD1P0, before entering low power modes the stepping for VDD1P0 must be disabled.*

## Detection Related Interfaces

- void [DCDC\\_GetDefaultDetectionConfig](#) (dcdc\_detection\_config\_t \*config)  
*Gets the default setting for detection configuration.*
- void [DCDC\\_SetDetectionConfig](#) (DCDC\_Type \*base, const dcdc\_detection\_config\_t \*config)  
*Configures the DCDC detection.*

## DCDC Miscellaneous Interfaces

- void [DCDC\\_SetClockSource](#) (DCDC\_Type \*base, dcdc\_clock\_source\_t clockSource)  
*Configures the DCDC clock source.*
- static void [DCDC\\_SetBandgapVoltageTrimValue](#) (DCDC\_Type \*base, uint32\_t trimValue)  
*Sets the bangap trim value(0~31) to trim bandgap voltage.*
- void [DCDC\\_GetDefaultLoopControlConfig](#) (dcdc\_loop\_control\_config\_t \*config)  
*Gets the default setting for loop control configuration.*
- void [DCDC\\_SetLoopControlConfig](#) (DCDC\_Type \*base, const dcdc\_loop\_control\_config\_t \*config)  
*Configures the DCDC loop control.*
- void [DCDC\\_SetInternalRegulatorConfig](#) (DCDC\_Type \*base, const dcdc\_internal\_regulator\_config\_t \*config)  
*Configures the DCDC internal regulator.*
- static void [DCDC\\_EnableAdjustDelay](#) (DCDC\_Type \*base, bool enable)  
*Adjusts delay to reduce ground noise.*

## DCDC Status Related Interfaces

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

## Application Guideline Interfaces

- void [DCDC\\_BootIntoCCM](#) (DCDC\_Type \*base)  
*Boots DCDC into CCM(continous conduction mode).*

## 5.2 Data Structure Documentation

### 5.2.1 struct dcdc\_config\_t

#### Data Fields

- dcdc\_control\_mode\_t controlMode  
*DCDC control mode.*
- dcdc\_trim\_input\_mode\_t trimInputModule  
*Hold trim input.*

### Field Documentation

- (1) `dcdc_control_mode_t dcdc_config_t::controlMode`
- (2) `dcdc_trim_input_mode_t dcdc_config_t::trimInputModule`

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

### Data Fields

- bool `enableUseHalfFreqForContinuous`  
*Set DCDC clock to half frequency for the continuous mode.*

### Field Documentation

- (1) `bool dcdc_min_power_config_t::enableUseHalfFreqForContinuous`

## 5.2.3 struct dcdc\_detection\_config\_t

### Data Fields

- bool `enableXtalokDetection`  
*Enable xtalok detection circuit.*
- bool `powerDownOverVoltageVdd1P8Detection`  
*Power down over-voltage detection comparator for VDD1P8.*
- bool `powerDownOverVoltageVdd1P0Detection`  
*Power down over-voltage detection comparator for VDD1P0.*
- bool `powerDownLowVoltageDetection`  
*Power down low-voltage detection comparator.*
- bool `powerDownOverCurrentDetection`  
*Power down over-current detection.*

**Field Documentation**

- (1) **bool dcdc\_detection\_config\_t::enableXtalokDetection**
- (2) **bool dcdc\_detection\_config\_t::powerDownOverVoltageVdd1P8Detection**
- (3) **bool dcdc\_detection\_config\_t::powerDownOverVoltageVdd1P0Detection**
- (4) **bool dcdc\_detection\_config\_t::powerDownLowVoltageDetection**
- (5) **bool dcdc\_detection\_config\_t::powerDownOverCurrentDetection**

**5.2.4 struct dc当地环控制配置\_t****Data Fields**

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

**Field Documentation**

- (1) **bool dc当地环控制配置\_t::enableCommonHysteresis**

This feature will improve transient supply ripple and efficiency.

- (2) **bool dc当地环控制配置\_t::enableCommonThresholdDetection**
- (3) **bool dc当地环控制配置\_t::enableDifferentialHysteresis**

This feature will improve transient supply ripple and efficiency.

- (4) `bool dcdc_loop_control_config_t::enableDifferentialThresholdDetection`
- (5) `bool dcdc_loop_control_config_t::enableInvertHysteresisSign`
- (6) `bool dcdc_loop_control_config_t::enableRCThresholdDetection`
- (7) `uint32_t dcdc_loop_control_config_t::enableRCscaleCircuit`

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

- (8) `uint32_t dcdc_loop_control_config_t::complementFeedForwardStep`

Two's complement feed forward step in duty cycle in the switching DC-DC converter. Each time this field makes a transition from 0x0, the loop filter of the DC-DC converter is stepped once by a value proportional to the change. This can be used to force a certain control loop behavior, such as improving response under known heavy load transients.

- (9) `uint32_t dcdc_loop_control_config_t::controlParameterMagnitude`

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

- (10) `uint32_t dcdc_loop_control_config_t::integralProportionalRatio`

## 5.2.5 struct dc当地内部调节器配置\_t

### Data Fields

- `uint32_t feedbackPoint`  
*Available range is 0~3.*

#### Field Documentation

- (1) `uint32_t dc当地内部调节器配置_t::feedbackPoint`

Select the feedback point of the internal regulator.

## 5.3 Macro Definition Documentation

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

Version 2.0.1.

### 5.3.2 #define VDD1P0\_TARGET\_VOLTAGE

#### Value:

{

```

 \
 600, 625, 650, 675, 700, 725, 750, 775, 800, 825, 850, 875, 900, 925, 950, 975, 1000, 1025, 1050,
1075, 1100, \
 \
 1125, 1150, 1175, 1200, 1225, 1250, 1275, 1300, 1325, 1350, 1375
 \
}

```

### 5.3.3 #define VDD1P8\_TARGET\_VOLTAGE

**Value:**

```

{
 \
 1500, 1525, 1550, 1575, 1600, 1625, 1650, 1675, 1700, 1725, 1750, 1775, 1800, 1825, 1850, 1875,
1900, 1925, \
 \
 1950, 1975, 2000, 2025, 2050, 2075, 2100, 2125, 2150, 2175, 2200, 2225, 2250, 2275
 \
}

```

## 5.4 Enumeration Type Documentation

### 5.4.1 enum \_dcdc\_status\_flags

Enumerator

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

### 5.4.2 enum dcdc\_core\_slice\_t

Enumerator

*kDCDC\_CORE0* CORE slice 0.  
*kDCDC\_CORE1* CORE slice 1.

### 5.4.3 enum dcde\_control\_mode\_t

Enumerator

*kDCDC\_SoftwareControl* Controlled by software.  
*kDCDC\_GPCControl* Controlled by GPC.

#### 5.4.4 enum dc当地 trim\_input\_mode\_t

Enumerator

*kDCDC\_SampleTrimInput* Sample trim input.

*kDCDC\_HoldTrimInput* Hold trim input.

#### 5.4.5 enum dc当地\_1P0\_target\_vol\_t

Enumerator

*kDCDC\_1P0Target0P6V* The target voltage value of VDD1P0 is 0.6V.

*kDCDC\_1P0Target0P625V* The target voltage value of VDD1P0 is 0.625V.

*kDCDC\_1P0Target0P65V* The target voltage value of VDD1P0 is 0.65V.

*kDCDC\_1P0Target0P675V* The target voltage value of VDD1P0 is 0.675V.

*kDCDC\_1P0Target0P7V* The target voltage value of VDD1P0 is 0.7V.

*kDCDC\_1P0Target0P725V* The target voltage value of VDD1P0 is 0.725V.

*kDCDC\_1P0Target0P75V* The target voltage value of VDD1P0 is 0.75V.

*kDCDC\_1P0Target0P775V* The target voltage value of VDD1P0 is 0.775V.

*kDCDC\_1P0Target0P8V* The target voltage value of VDD1P0 is 0.8V.

*kDCDC\_1P0Target0P825V* The target voltage value of VDD1P0 is 0.825V.

*kDCDC\_1P0Target0P85V* The target voltage value of VDD1P0 is 0.85V.

*kDCDC\_1P0Target0P875V* The target voltage value of VDD1P0 is 0.875V.

*kDCDC\_1P0Target0P9V* The target voltage value of VDD1P0 is 0.9V.

*kDCDC\_1P0Target0P925V* The target voltage value of VDD1P0 is 0.925V.

*kDCDC\_1P0Target0P95V* The target voltage value of VDD1P0 is 0.95V.

*kDCDC\_1P0Target0P975V* The target voltage value of VDD1P0 is 0.975V.

*kDCDC\_1P0Target1P0V* The target voltage value of VDD1P0 is 1.0V.

*kDCDC\_1P0Target1P025V* The target voltage value of VDD1P0 is 1.025V.

*kDCDC\_1P0Target1P05V* The target voltage value of VDD1P0 is 1.05V.

*kDCDC\_1P0Target1P075V* The target voltage value of VDD1P0 is 1.075V.

*kDCDC\_1P0Target1P1V* The target voltage value of VDD1P0 is 1.1V.

*kDCDC\_1P0Target1P125V* The target voltage value of VDD1P0 is 1.125V.

*kDCDC\_1P0Target1P15V* The target voltage value of VDD1P0 is 1.15V.

*kDCDC\_1P0Target1P175V* The target voltage value of VDD1P0 is 1.175V.

*kDCDC\_1P0Target1P2V* The target voltage value of VDD1P0 is 1.2V.

*kDCDC\_1P0Target1P225V* The target voltage value of VDD1P0 is 1.225V.

*kDCDC\_1P0Target1P25V* The target voltage value of VDD1P0 is 1.25V.

*kDCDC\_1P0Target1P275V* The target voltage value of VDD1P0 is 1.275V.

*kDCDC\_1P0Target1P3V* The target voltage value of VDD1P0 is 1.3V.

*kDCDC\_1P0Target1P325V* The target voltage value of VDD1P0 is 1.325V.

*kDCDC\_1P0Target1P35V* The target voltage value of VDD1P0 is 1.35V.

*kDCDC\_1P0Target1P375V* The target voltage value of VDD1P0 is 1.375V.

## 5.4.6 enum dc当地\_1P8\_target\_vol\_t

Enumerator

*kDCDC\_1P8Target1P5V* The target voltage value of VDD1P8 is 1.5V.  
*kDCDC\_1P8Target1P525V* The target voltage value of VDD1P8 is 1.525V.  
*kDCDC\_1P8Target1P55V* The target voltage value of VDD1P8 is 1.55V.  
*kDCDC\_1P8Target1P575V* The target voltage value of VDD1P8 is 1.575V.  
*kDCDC\_1P8Target1P6V* The target voltage value of VDD1P8 is 1.6V.  
*kDCDC\_1P8Target1P625V* The target voltage value of VDD1P8 is 1.625V.  
*kDCDC\_1P8Target1P65V* The target voltage value of VDD1P8 is 1.65V.  
*kDCDC\_1P8Target1P675V* The target voltage value of VDD1P8 is 1.675V.  
*kDCDC\_1P8Target1P7V* The target voltage value of VDD1P8 is 1.7V.  
*kDCDC\_1P8Target1P725V* The target voltage value of VDD1P8 is 1.725V.  
*kDCDC\_1P8Target1P75V* The target voltage value of VDD1P8 is 1.75V.  
*kDCDC\_1P8Target1P775V* The target voltage value of VDD1P8 is 1.775V.  
*kDCDC\_1P8Target1P8V* The target voltage value of VDD1P8 is 1.8V.  
*kDCDC\_1P8Target1P825V* The target voltage value of VDD1P8 is 1.825V.  
*kDCDC\_1P8Target1P85V* The target voltage value of VDD1P8 is 1.85V.  
*kDCDC\_1P8Target1P875V* The target voltage value of VDD1P8 is 1.875V.  
*kDCDC\_1P8Target1P9V* The target voltage value of VDD1P8 is 1.9V.  
*kDCDC\_1P8Target1P925V* The target voltage value of VDD1P8 is 1.925V.  
*kDCDC\_1P8Target1P95V* The target voltage value of VDD1P8 is 1.95V.  
*kDCDC\_1P8Target1P975V* The target voltage value of VDD1P8 is 1.975V.  
*kDCDC\_1P8Target2P0V* The target voltage value of VDD1P8 is 2.0V.  
*kDCDC\_1P8Target2P025V* The target voltage value of VDD1P8 is 2.025V.  
*kDCDC\_1P8Target2P05V* The target voltage value of VDD1P8 is 2.05V.  
*kDCDC\_1P8Target2P075V* The target voltage value of VDD1P8 is 2.075V.  
*kDCDC\_1P8Target2P1V* The target voltage value of VDD1P8 is 2.1V.  
*kDCDC\_1P8Target2P125V* The target voltage value of VDD1P8 is 2.125V.  
*kDCDC\_1P8Target2P15V* The target voltage value of VDD1P8 is 2.15V.  
*kDCDC\_1P8Target2P175V* The target voltage value of VDD1P8 is 2.175V.  
*kDCDC\_1P8Target2P2V* The target voltage value of VDD1P8 is 2.2V.  
*kDCDC\_1P8Target2P225V* The target voltage value of VDD1P8 is 2.225V.  
*kDCDC\_1P8Target2P25V* The target voltage value of VDD1P8 is 2.25V.  
*kDCDC\_1P8Target2P275V* The target voltage value of VDD1P8 is 2.275V.

## 5.4.7 enum dc当地\_comparator\_current\_bias\_t

Enumerator

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

***kDCDC\_ComparatorCurrentBias400nA*** The current bias of low power comparator is 400nA.

#### 5.4.8 enum dc当地\_c\_peak\_current\_threshold\_t

Enumerator

***kDCDC\_PeakCurrentRunMode250mA*** Over peak current threshold in low power mode is 250mA, in run mode is 1.5A.

***kDCDC\_PeakCurrentRunMode200mA*** Over peak current threshold in low power mode is 200mA, in run mode is 1.5A.

***kDCDC\_PeakCurrentRunMode250mA*** Over peak current threshold in low power mode is 250mA, in run mode is 2A.

***kDCDC\_PeakCurrentRunMode200mA*** Over peak current threshold in low power mode is 200mA, in run mode is 2A.

#### 5.4.9 enum dc当地\_clock\_source\_t

Enumerator

***kDCDC\_ClockAutoSwitch*** Automatic clock switch from internal oscillator to external clock.

***kDCDC\_ClockInternalOsc*** Use internal oscillator.

***kDCDC\_ClockExternalOsc*** Use external 24M crystal oscillator.

#### 5.4.10 enum dc当地\_voltage\_output\_sel\_t

Enumerator

***kDCDC\_VoltageOutput1P8*** 1.8V output.

***kDCDC\_VoltageOutput1P0*** 1.0V output.

### 5.5 Function Documentation

#### 5.5.1 void DCDC\_Init ( DCDC\_Type \* *base*, const dc当地\_config\_t \* *config* )

Parameters

---

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | DCDC peripheral base address.                           |
| <i>config</i> | Pointer to the <a href="#">dcdc_config_t</a> structure. |

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

Parameters

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

### 5.5.3 void DCDC\_GetDefaultConfig ( dcdc\_config\_t \* *config* )

This function initializes the user configuration structure to a default value. The default values are:

```
* config->controlMode = kDCDC_StaticControl;
* config->trimInputMode = kDCDC_SampleTrimInput;
* config->enableDcdcTimeout = false;
* config->enableSwitchingConverterOutput = false;
*
```

Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>config</i> | Pointer to configuration structure. See to <a href="#">dcdc_config_t</a> . |
|---------------|----------------------------------------------------------------------------|

### 5.5.4 static uint16\_t DCDC\_GetVDD1P0LowPowerModeTargetVoltage ( DCDC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The VDD1P0's voltage value in low power mode and the unit is "mV".

### 5.5.5 static void DCDC\_SetVDD1P0BuckModeTargetVoltage ( DCDC\_Type \* *base*, dcdc\_core\_slice\_t *core*, dcdc\_1P0\_target\_vol\_t *targetVoltage* ) [inline], [static]

Parameters

|                      |                                                                                      |
|----------------------|--------------------------------------------------------------------------------------|
| <i>base</i>          | DCDC peripheral base address.                                                        |
| <i>core</i>          | Core for DCDC to control.                                                            |
| <i>targetVoltage</i> | The target value of VDD1P0 in buck mode, see <a href="#">dcdc_1P0_target_vol_t</a> . |

### 5.5.6 static uint16\_t DCDC\_GetVDD1P0BuckModeTargetVoltage ( DCDC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The VDD1P0's voltage value in buck mode and the unit is "mV".

### 5.5.7 static void DCDC\_GPC\_SetVDD1P0LowPowerModeTargetVoltage ( DCDC\_Type \* *base*, dcdc\_core\_slice\_t *core*, dcdc\_1P0\_target\_vol\_t *targetVoltage* ) [inline], [static]

Parameters

|                      |                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------|
| <i>base</i>          | DCDC peripheral base address.                                                             |
| <i>core</i>          | Core for DCDC to control.                                                                 |
| <i>targetVoltage</i> | The target value of VDD1P0 in low power mode, see <a href="#">dcdc_1P0_target_vol_t</a> . |

### 5.5.8 static void DCDC\_GPC\_EnableVDD1P0LowPowerMode ( DCDC\_Type \* *base*, dcdc\_core\_slice\_t *core*, bool *enable* ) [inline], [static]

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DCDC peripheral base address. |
| <i>core</i>   | Core for DCDC to control.     |
| <i>enable</i> | Enable the output or not.     |

**5.5.9 static void DCDC\_SetVDD1P8TargetVoltage ( DCDC\_Type \* *base*, dcdc\_core\_slice\_t *core*, dcdc\_1P8\_target\_vol\_t *targetVoltage* ) [inline], [static]**

Parameters

|                      |                                                                         |
|----------------------|-------------------------------------------------------------------------|
| <i>base</i>          | DCDC peripheral base address.                                           |
| <i>core</i>          | Core for DCDC to control.                                               |
| <i>targetVoltage</i> | The target value of VDD1P8, see <a href="#">dcdc_1P8_target_vol_t</a> . |

**5.5.10 static uint16\_t DCDC\_GetVDD1P8TargetVoltage ( DCDC\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The VDD1P8's voltage value and the unit is "mV".

**5.5.11 static void DCDC\_EnableVDD1P0TargetVoltageStepping ( DCDC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

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

|               |                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enable</i> | Used to control the behavior. <ul style="list-style-type: none"> <li>• <b>true</b> Enables stepping for VDD1P0.</li> <li>• <b>false</b> Disables stepping for VDD1P0.</li> </ul> |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

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

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

```
* config->enableXtalokDetection = false;
* config->powerDownOverVoltageVdd1P8Detection = true;
* config->powerDownOverVoltageVdd1P0Detection = true;
* config->powerDownLowVoltageDetection = false;
* config->powerDownOverCurrentDetection = true;
* config->powerDownPeakCurrentDetection = true;
* config->powerDownZeroCrossDetection = true;
* config->OverCurrentThreshold = kDCDC_OverCurrentThresholdAlt0;
* config->PeakCurrentThreshold = kDCDC_PeakCurrentThresholdAlt0;
*
```

Parameters

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <i>config</i> | Pointer to configuration structure. See to <a href="#">dcdc_detection_config_t</a> . |
|---------------|--------------------------------------------------------------------------------------|

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

Parameters

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <i>base</i>   | DCDC peripheral base address.                                                        |
| <i>config</i> | Pointer to configuration structure. See to <a href="#">dcdc_detection_config_t</a> . |

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

Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>base</i>        | DCDC peripheral base address.                                       |
| <i>clockSource</i> | Clock source for DCDC. See to <a href="#">dcdc_clock_source_t</a> . |

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

Parameters

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

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

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

```
* config->enableCommonHysteresis = false;
* config->enableCommonThresholdDetection = false;
* config->enableInvertHysteresisSign = false;
* config->enableRCThresholdDetection = false;
* config->enableRCScaleCircuit = 0U;
* config->complementFeedForwardStep = 0U;
* config->controlParameterMagnitude = 2U;
* config->integralProportionalRatio = 2U;
*
```

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>config</i> | Pointer to configuration structure. See to <a href="#">dcdc_loop_control_config_t</a> . |
|---------------|-----------------------------------------------------------------------------------------|

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

Parameters

---

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | DCDC peripheral base address.                                                           |
| <i>config</i> | Pointer to configuration structure. See to <a href="#">dcdc_loop_control_config_t</a> . |

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

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | DCDC peripheral base address.                                                                 |
| <i>config</i> | Pointer to configuration structure. See to <a href="#">dcdc_internal_regulator_config_t</a> . |

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

Parameters

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

### 5.5.20 static uint32\_t DCDC\_GetStatusFlags ( DCDC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Mask of asserted status flags. See to [\\_dcdc\\_status\\_flags](#).

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

```
* pwd_zcd=0x1;
* pwd_cmp_offset=0x0;
* dcdc_loopctrl_en_rcscale=0x3;
*
```

### Parameters

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

# Chapter 6

## ROMAPI Driver

### 6.1 Overview

The ROMAPI driver provides the functionalities to operate the external NOR Flash connected to the FLEXSPI controller.

The ROMAPI driver supports:

- Initialize serial NOR flash via FLEXSPI
- Program data to serial NOR flash via FLEXSPI.
- Erase serial NOR flash region via FLEXSPI.

### Data Structures

- struct `serial_nor_config_option_t`  
*Serial NOR Configuration Option.* [More...](#)
- struct `flexspi_lut_seq_t`  
*FLEXSPI LUT Sequence structure.* [More...](#)
- struct `flexspi_dll_time_t`  
*FLEXSPI DLL time.* [More...](#)
- struct `flexspi_mem_config_t`  
*FLEXSPI Memory Configuration Block.* [More...](#)
- struct `flexspi_nor_config_t`  
*Serial NOR configuration block.* [More...](#)
- struct `flexspi_xfer_t`  
*FLEXSPI Transfer Context.* [More...](#)

### Macros

- #define `FSL_ROM_ROMAPI_VERSION` (`MAKE_VERSION(1U, 1U, 2U)`)  
*ROM API version 1.1.2.*
- #define `FSL_ROM_FLEXSPINOR_DRIVER_VERSION` (`MAKE_VERSION(1U, 7U, 0U)`)  
*ROM FLEXSPI NOR driver version 1.7.0.*
- #define `kROM_StatusGroup_FLEXSPINOR` 201U  
*ROM FLEXSPI NOR status group number.*
- #define `FSL_ROM_FLEXSPI_BITMASK`(bit\_offset) (`1U << (bit_offset)`)  
*Generate bit mask.*
- #define `FLEXSPI_CFG_BLK_TAG` (0x42464346UL)  
*FLEXSPI memory config block related definitions.*
- #define `FLEXSPI_CFG_BLK_VERSION` (0x56010400UL)  
*V1.4.0.*
- #define `NOR_CMD_LUT_SEQ_IDX_READ` 0U  
*NOR LUT sequence index used for default LUT assignment.*
- #define `NOR_CMD_LUT_SEQ_IDX_READSTATUS` 1U  
*Read Status LUT sequence id in lookupTable stored in config block.*

- #define NOR\_CMD\_LUT\_SEQ\_IDX\_READSTATUS\_XPI 2U  
*Read status DPI/QPI/OPI sequence id in lookupTable stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_WRITEENABLE 3U  
*Write Enable sequence id in lookupTable stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_WRITEENABLE\_XPI 4U  
*Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_ERASESECTOR 5U  
*Erase Sector sequence id in lookupTable stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_ERASEBLOCK 8U  
*Erase Block sequence id in lookupTable stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_PAGEPROGRAM 9U  
*Program sequence id in lookupTable stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_CHIPERASE 11U  
*Chip Erase sequence in lookupTable id stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_READ\_SFDP 13U  
*Read SFDP sequence in lookupTable id stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_RESTORE\_NOCMD 14U  
*Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block.*
- #define NOR\_CMD\_LUT\_SEQ\_IDX\_EXIT\_NOCMD 15U  
*Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block.*
- #define MISRA\_CAST(to\_type, to\_var, from\_type, from\_var)  
*convert the type for MISRA*

## Enumerations

- enum {
   
kSerialFlash\_ISSI\_ManufacturerID = 0x9DU,  
kSerialFlash\_Adesto\_ManufacturerID = 0x1F,  
kSerialFlash\_Winbond\_ManufacturerID = 0xEFU,  
kSerialFlash\_Cypress\_ManufacturerID = 0x01U }  
*Manufacturer ID.*
- enum \_flexspi\_nor\_status {
   
kStatus\_ROM\_FLEXSPI\_SequenceExecutionTimeout,  
kStatus\_ROM\_FLEXSPI\_InvalidSequence = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 1),  
kStatus\_ROM\_FLEXSPI\_DeviceTimeout = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 2),  
kStatus\_ROM\_FLEXSPINOR\_SFDP\_NotFound,  
kStatus\_ROM\_FLEXSPINOR\_Flash\_NotFound,  
kStatus\_FLEXSPINOR\_DTRRead\_DummyProbeFailed }  
*ROM FLEXSPI NOR flash status.*
- enum flexspi\_operation\_t {
   
kFLEXSPIOperation\_Command,  
kFLEXSPIOperation\_Config,  
kFLEXSPIOperation\_Write,  
kFLEXSPIOperation\_Read }  
*FLEXSPI Operation Context.*

## Functions

- void ROM\_API\_Init (void)

*ROM API init.*

## Common ROMAPI features info defines

- #define **FSL\_ROM\_HAS\_FLEXSPINOR\_API** (1)  
*ROM has FLEXSPI NOR API.*
- #define **FSL\_ROM\_HAS\_RUNBOOTLOADER\_API** (1)  
*ROM has run bootloader API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_GET\_CONFIG** (1)  
*ROM has FLEXSPI NOR get config API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_FLASH\_INIT** (1)  
*ROM has flash init API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE** (1)  
*ROM has erase API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE\_SECTOR** (1)  
*ROM has erase sector API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE\_BLOCK** (1)  
*ROM has erase block API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_ERASE\_ALL** (1)  
*ROM has erase all API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_READ** (1)  
*ROM has read API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_UPDATE\_LUT** (1)  
*ROM has update lut API.*
- #define **FSL\_ROM\_FLEXSPINOR\_API\_HAS\_FEATURE\_CMD\_XFER** (1)  
*ROM has FLEXSPI command API.*

## Support for init FLEXSPI NOR configuration

- enum {
 **kSerialFlash\_1Pad** = 1U,  
**kSerialFlash\_2Pads** = 2U,  
**kSerialFlash\_4Pads** = 4U,  
**kSerialFlash\_8Pads** = 8U }
   
*Flash Pad Definitions.*
- enum {
 **kFLEXSPIClk\_SDR**,  
**kFLEXSPIClk\_DDR** }
   
*FLEXSPI clock configuration type.*
- enum **\_flexspi\_read\_sample\_clk** {
 **kFLEXSPIReadSampleClk\_LoopbackInternally** = 0U,  
**kFLEXSPIReadSampleClk\_LoopbackFromDqsPad** = 1U,  
**kFLEXSPIReadSampleClk\_LoopbackFromSckPad** = 2U,  
**kFLEXSPIReadSampleClk\_ExternalInputFromDqsPad** = 3U }
   
*FLEXSPI Read Sample Clock Source definition.*
- enum { **kFLEXSPIDeviceType\_SerialNOR** = 1U }
   
*Flash Type Definition.*
- enum {

```
kDeviceConfigCmdType_Generic,
kDeviceConfigCmdType_QuadEnable,
kDeviceConfigCmdType_Spi2Xpi,
kDeviceConfigCmdType_Xpi2Spi,
kDeviceConfigCmdType_Spi2NoCmd,
kDeviceConfigCmdType_Reset }
```

*Flash Configuration Command Type.*

- enum `_flexspi_serial_clk_freq` {
 `kFLEXSPISerialClk_NoChange` = 0U,
 `kFLEXSPISerialClk_30MHz` = 1U,
 `kFLEXSPISerialClk_50MHz` = 2U,
 `kFLEXSPISerialClk_60MHz` = 3U,
 `kFLEXSPISerialClk_75MHz` = 4U,
 `kFLEXSPISerialClk_80MHz` = 5U,
 `kFLEXSPISerialClk_100MHz` = 6U,
 `kFLEXSPISerialClk_133MHz` = 7U,
 `kFLEXSPISerialClk_166MHz` = 8U }

*Definitions for FLEXSPI Serial Clock Frequency.*

- enum {
 `kFLEXSPIMiscOffset_DiffClkEnable` = 0U,
 `kFLEXSPIMiscOffset_Ck2Enable` = 1U,
 `kFLEXSPIMiscOffset_ParallelEnable` = 2U,
 `kFLEXSPIMiscOffset_WordAddressableEnable` = 3U,
 `kFLEXSPIMiscOffset_SafeConfigFreqEnable` = 4U,
 `kFLEXSPIMiscOffset_PadSettingOverrideEnable` = 5U,
 `kFLEXSPIMiscOffset_DdrModeEnable` = 6U,
 `kFLEXSPIMiscOffset_UseValidTimeForAllFreq` = 7U }

*Misc feature bit definitions.*

## Enter Bootloader

- void `ROM_RunBootloader` (void \*arg)

*Enter Bootloader.*

## GetConfig

- `status_t ROM_FLEXSPI_NorFlash_GetConfig` (uint32\_t instance, `flexspi_nor_config_t` \*config, `serial_nor_config_option_t` \*option)

*Get FLEXSPI NOR Configuration Block based on specified option.*

## Initialization

- `status_t ROM_FLEXSPI_NorFlash_Init` (uint32\_t instance, `flexspi_nor_config_t` \*config)

*Initialize Serial NOR devices via FLEXSPI.*

## Programming

- `status_t ROM_FLEXSPI_NorFlash_ProgramPage` (`uint32_t` instance, `flexspi_nor_config_t` \*config, `uint32_t` dst\_addr, const `uint32_t` \*src)  
*Program data to Serial NOR via FLEXSPI.*

## Reading

- `status_t ROM_FLEXSPI_NorFlash_Read` (`uint32_t` instance, `flexspi_nor_config_t` \*config, `uint32_t` \*dst, `uint32_t` start, `uint32_t` lengthInBytes)  
*Read data from Serial NOR via FLEXSPI.*

## Erasing

- `status_t ROM_FLEXSPI_NorFlash_Erase` (`uint32_t` instance, `flexspi_nor_config_t` \*config, `uint32_t` start, `uint32_t` length)  
*Erase Flash Region specified by address and length.*
- `status_t ROM_FLEXSPI_NorFlash_EraseSector` (`uint32_t` instance, `flexspi_nor_config_t` \*config, `uint32_t` start)  
*Erase one sector specified by address.*
- `status_t ROM_FLEXSPI_NorFlash_EraseBlock` (`uint32_t` instance, `flexspi_nor_config_t` \*config, `uint32_t` start)  
*Erase one block specified by address.*
- `status_t ROM_FLEXSPI_NorFlash_EraseAll` (`uint32_t` instance, `flexspi_nor_config_t` \*config)  
*Erase all the Serial NOR devices connected on FLEXSPI.*

## Command

- `status_t ROM_FLEXSPI_NorFlash_CommandXfer` (`uint32_t` instance, `flexspi_xfer_t` \*xfer)  
*FLEXSPI command.*

## UpdateLut

- `status_t ROM_FLEXSPI_NorFlash_UpdateLut` (`uint32_t` instance, `uint32_t` seqIndex, const `uint32_t` \*lutBase, `uint32_t` seqNumber)  
*Configure FLEXSPI Lookup table.*

## Device status

- `status_t ROM_FLEXSPI_NorFlash_WaitBusy` (`uint32_t` instance, `flexspi_nor_config_t` \*config, `bool` isParallelMode, `uint32_t` address)  
*Wait until device is idle.*

## ClearCache

- `void ROM_FLEXSPI_NorFlash_ClearCache` (`uint32_t` instance)  
*Software reset for the FLEXSPI logic.*

## 6.2 Data Structure Documentation

### 6.2.1 struct serial\_nor\_config\_option\_t

### 6.2.2 struct flexspi\_lut\_seq\_t

#### Data Fields

- `uint8_t seqNum`  
*Sequence Number, valid number: 1-16.*
- `uint8_t seqId`  
*Sequence Index, valid number: 0-15.*

### 6.2.3 struct flexspi\_dll\_time\_t

#### Data Fields

- `uint8_t time_100ps`  
*Data valid time, in terms of 100ps.*
- `uint8_t delay_cells`  
*Data valid time, in terms of delay cells.*

### 6.2.4 struct flexspi\_mem\_config\_t

#### Data Fields

- `uint32_t tag`  
*[0x000-0x003] Tag, fixed value 0x42464346UL*
- `uint32_t version`  
*[0x004-0x007] Version,[31:24] -'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix*
- `uint32_t reserved0`  
*[0x008-0x00b] Reserved for future use*
- `uint8_t readSampleClkSrc`  
*[0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3*
- `uint8_t csHoldTime`  
*[0x00d-0x00d] Data hold time, default value: 3*
- `uint8_t csSetupTime`  
*[0x00e-0x00e] Date setup time, default value: 3*
- `uint8_t columnAddressWidth`  
*[0x00f-0x00f] Column Address width, for HyperBus protocol, it is fixed to 3, For Serial NAND, need to refer to datasheet*
- `uint8_t deviceModeCfgEnable`  
*[0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable*
- `uint8_t deviceModeType`  
*[0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch, Generic configuration, etc.*

- **uint16\_t waitTimeCfgCommands**  
`[0x012-0x013]` Wait time for all configuration commands, unit: 100us, Used for DPI/QPI/OPI switch or reset command
- **flexspi\_lut\_seq\_t deviceModeSeq**  
`[0x014-0x017]` Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUT sequence number, [31:16] Reserved
- **uint32\_t deviceModeArg**  
`[0x018-0x01b]` Argument/Parameter for device configuration
- **uint8\_t configCmdEnable**  
`[0x01c-0x01c]` Configure command Enable Flag, 1 - Enable, 0 - Disable
- **uint8\_t configModeType [3]**  
`[0x01d-0x01f]` Configure Mode Type, similar as deviceModeType
- **flexspi\_lut\_seq\_t configCmdSeqs [3]**  
`[0x020-0x02b]` Sequence info for Device Configuration command, similar as deviceModeSeq
- **uint32\_t reserved1**  
`[0x02c-0x02f]` Reserved for future use
- **uint32\_t configCmdArgs [3]**  
`[0x030-0x03b]` Arguments/Parameters for device Configuration commands
- **uint32\_t reserved2**  
`[0x03c-0x03f]` Reserved for future use
- **uint32\_t controllerMiscOption**  
`[0x040-0x043]` Controller Misc Options, see Misc feature bit definitions for more details
- **uint8\_t deviceType**  
`[0x044-0x044]` Device Type: See Flash Type Definition for more details
- **uint8\_t sflashPadType**  
`[0x045-0x045]` Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal
- **uint8\_t serialClkFreq**  
`[0x046-0x046]` Serial Flash Frequency, device specific definitions.
- **uint8\_t lutCustomSeqEnable**  
`[0x047-0x047]` LUT customization Enable, it is required if the program/erase cannot be done using 1 LUT sequence, currently, only applicable to HyperFLASH
- **uint32\_t reserved3 [2]**  
`[0x048-0x04f]` Reserved for future use
- **uint32\_t sflashA1Size**  
`[0x050-0x053]` Size of Flash connected to A1
- **uint32\_t sflashA2Size**  
`[0x054-0x057]` Size of Flash connected to A2
- **uint32\_t sflashB1Size**  
`[0x058-0x05b]` Size of Flash connected to B1
- **uint32\_t sflashB2Size**  
`[0x05c-0x05f]` Size of Flash connected to B2
- **uint32\_t csPadSettingOverride**  
`[0x060-0x063]` CS pad setting override value
- **uint32\_t sclkPadSettingOverride**  
`[0x064-0x067]` SCK pad setting override value
- **uint32\_t dataPadSettingOverride**  
`[0x068-0x06b]` data pad setting override value
- **uint32\_t dqsPadSettingOverride**  
`[0x06c-0x06f]` DQS pad setting override value
- **uint32\_t timeoutInMs**  
`[0x070-0x073]` Timeout threshold for read status command

- `uint32_t commandInterval`  
`[0x074-0x077] CS deselect interval between two commands`
- `flexspi_dll_time_t dataValidTime [2]`  
`[0x078-0x07b] CLK edge to data valid time for PORT A and PORT B`
- `uint16_t busyOffset`  
`[0x07c-0x07d] Busy offset, valid value: 0-31`
- `uint16_t busyBitPolarity`  
`[0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 - busy flag is 0 when flash device is busy`
- `uint32_t lookupTable [64]`  
`[0x080-0x17f] Lookup table holds Flash command sequences`
- `flexspi_lut_seq_t lutCustomSeq [12]`  
`[0x180-0x1af] Customizable LUT Sequences`
- `uint32_t reserved4 [4]`  
`[0x1b0-0x1bf] Reserved for future use`

### Field Documentation

(1) `uint8_t flexspi_mem_config_t::deviceModeType`

(2) `uint8_t flexspi_mem_config_t::serialClkFreq`

See System Boot Chapter for more details

### 6.2.5 struct `flexspi_nor_config_t`

#### Data Fields

- `flexspi_mem_config_t memConfig`  
`Common memory configuration info via FLEXSPI.`
- `uint32_t pageSize`  
`Page size of Serial NOR.`
- `uint32_t sectorSize`  
`Sector size of Serial NOR.`
- `uint8_t ipcCmdSerialClkFreq`  
`Clock frequency for IP command.`
- `uint8_t isUniformBlockSize`  
`Sector/Block size is the same.`
- `uint8_t isDataOrderSwapped`  
`Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)`
- `uint8_t reserved0 [1]`  
`Reserved for future use.`
- `uint8_t serialNorType`  
`Serial NOR Flash type: 0/1/2/3.`
- `uint8_t needExitNoCmdMode`  
`Need to exit NoCmd mode before other IP command.`
- `uint8_t halfClkForNonReadCmd`  
`Half the Serial Clock for non-read command: true/false.`
- `uint8_t needRestoreNoCmdMode`

- `uint32_t blockSize`  
*Block size.*
- `uint32_t reserve2 [11]`  
*Reserved for future use.*

## 6.2.6 struct flexspi\_xfer\_t

### Data Fields

- `flexspi_operation_t operation`  
*FLEXSPI operation.*
- `uint32_t baseAddress`  
*FLEXSPI operation base address.*
- `uint32_t seqId`  
*Sequence Id.*
- `uint32_t seqNum`  
*Sequence Number.*
- `bool isParallelModeEnable`  
*Is a parallel transfer.*
- `uint32_t * txBuffer`  
*Tx buffer.*
- `uint32_t txSize`  
*Tx size in bytes.*
- `uint32_t * rxBuffer`  
*Rx buffer.*
- `uint32_t rxSize`  
*Rx size in bytes.*

## 6.3 Macro Definition Documentation

- 6.3.1 `#define FSL_ROM_ROMAPI_VERSION (MAKE_VERSION(1U, 1U, 2U))`
- 6.3.2 `#define FSL_ROM_FLEXSPINOR_DRIVER_VERSION (MAKE_VERSION(1U, 7U, 0U))`
- 6.3.3 `#define FSL_ROM_HAS_FLEXSPINOR_API (1)`
- 6.3.4 `#define FSL_ROM_HAS_RUNBOOTLOADER_API (1)`
- 6.3.5 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_GET_CONFIG (1)`
- 6.3.6 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_FLASH_INIT (1)`
- 6.3.7 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE (1)`
- 6.3.8 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_SECTOR (1)`
- 6.3.9 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_BLOCK (1)`
- 6.3.10 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_ALL (1)`
- 6.3.11 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_READ (1)`
- 6.3.12 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_UPDATE_LUT (1)`
- 6.3.13 `#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_CMD_XFER (1)`
- 6.3.14 `#define kROM_StatusGroup_FLEXSPINOR 201U`  
ascii "FCFB" Big Endian
- 6.3.15 `#define FLEXSPI_CFG_BLK_TAG (0x42464346UL)`
- 6.3.16 `#define NOR_CMD_LUT_SEQ_IDX_READ 0U`

Note

It will take effect if the lut sequences are not customized.READ LUT sequence id in lookupTable stored in config block

## 6.4 Enumeration Type Documentation

### 6.4.1 anonymous enum

Enumerator

*kSerialFlash\_1Pad* 1-wire communication  
*kSerialFlash\_2Pads* 2-wire communication  
*kSerialFlash\_4Pads* 4-wire communication  
*kSerialFlash\_8Pads* 8-wire communication

### 6.4.2 anonymous enum

Enumerator

*kFLEXSPIClk\_SDR* Clock configure for SDR mode.  
*kFLEXSPIClk\_DDR* Clock configure for DDR mode.

### 6.4.3 enum \_flexspi\_read\_sample\_clk

Enumerator

*kFLEXSPIReadSampleClk\_LoopbackInternally* FLEXSPI Read Sample Clock Source from the Internal loopback.  
*kFLEXSPIReadSampleClk\_LoopbackFromDqsPad* FLEXSPI Read Sample Clock Source from the Dqs Pad loopback.  
*kFLEXSPIReadSampleClk\_LoopbackFromSckPad* FLEXSPI Read Sample Clock Source from the Sck Pad loopback.  
*kFLEXSPIReadSampleClk\_ExternalInputFromDqsPad* FLEXSPI Read Sample Clock Source from the External Input by the Dqs Pad.

### 6.4.4 anonymous enum

Enumerator

*kFLEXSPIDeviceType\_SerialNOR* Flash device is Serial NOR.

#### 6.4.5 anonymous enum

Enumerator

- kDeviceConfigCmdType\_Generic*** Generic command, for example: configure dummy cycles, drive strength, etc.
- kDeviceConfigCmdType\_QuadEnable*** Quad Enable command.
- kDeviceConfigCmdType\_Spi2Xpi*** Switch from SPI to DPI/QPI/OPI mode.
- kDeviceConfigCmdType\_Xpi2Spi*** Switch from DPI/QPI/OPI to SPI mode.
- kDeviceConfigCmdType\_Spi2NoCmd*** Switch to 0-4-4/0-8-8 mode.
- kDeviceConfigCmdType\_Reset*** Reset device command.

#### 6.4.6 enum \_flexspi\_serial\_clk\_freq

Enumerator

- kFLEXSPISerialClk\_NoChange*** FlexSPI serial clock no changed.
- kFLEXSPISerialClk\_30MHz*** FlexSPI serial clock 30MHz.
- kFLEXSPISerialClk\_50MHz*** FlexSPI serial clock 50MHz.
- kFLEXSPISerialClk\_60MHz*** FlexSPI serial clock 60MHz.
- kFLEXSPISerialClk\_75MHz*** FlexSPI serial clock 75MHz.
- kFLEXSPISerialClk\_80MHz*** FlexSPI serial clock 80MHz.
- kFLEXSPISerialClk\_100MHz*** FlexSPI serial clock 100MHz.
- kFLEXSPISerialClk\_133MHz*** FlexSPI serial clock 133MHz.
- kFLEXSPISerialClk\_166MHz*** FlexSPI serial clock 166MHz.

#### 6.4.7 anonymous enum

Enumerator

- kFLEXSPIMiscOffset\_DiffClkEnable*** Bit for Differential clock enable.
- kFLEXSPIMiscOffset\_Ck2Enable*** Bit for CK2 enable.
- kFLEXSPIMiscOffset\_ParallelEnable*** Bit for Parallel mode enable.
- kFLEXSPIMiscOffset\_WordAddressableEnable*** Bit for Word Addressable enable.
- kFLEXSPIMiscOffset\_SafeConfigFreqEnable*** Bit for Safe Configuration Frequency enable.
- kFLEXSPIMiscOffset\_PadSettingOverrideEnable*** Bit for Pad setting override enable.
- kFLEXSPIMiscOffset\_DdrModeEnable*** Bit for DDR clock configuration indication.
- kFLEXSPIMiscOffset\_UseValidTimeForAllFreq*** Bit for DLLCR settings under all modes.

#### 6.4.8 anonymous enum

Enumerator

- kSerialFlash\_ISSI\_ManufacturerID*** Manufacturer ID of the ISSI serial flash.

*kSerialFlash\_Adesto\_ManufacturerID* Manufacturer ID of the Adesto Technologies serial flash.

*kSerialFlash\_Winbond\_ManufacturerID* Manufacturer ID of the Winbond serial flash.

*kSerialFlash\_Cypress\_ManufacturerID* Manufacturer ID for Cypress.

### 6.4.9 enum \_flexspi\_nor\_status

Enumerator

*kStatus\_ROM\_FLEXSPI\_SequenceExecutionTimeout* Status for Sequence Execution timeout.

*kStatus\_ROM\_FLEXSPI\_InvalidSequence* Status for Invalid Sequence.

*kStatus\_ROM\_FLEXSPI\_DeviceTimeout* Status for Device timeout.

*kStatus\_ROM\_FLEXSPINOR\_SFDP\_NotFound* Status for SFDP read failure.

*kStatus\_ROM\_FLEXSPINOR\_Flash\_NotFound* Status for Flash detection failure.

*kStatus\_FLEXSPINOR\_DTRRead\_DummyProbeFailed* Status for DDR Read dummy probe failure.

### 6.4.10 enum flexspi\_operation\_t

Enumerator

*kFLEXSPIOperation\_Command* FLEXSPI operation: Only command, both TX and RX buffer are ignored.

*kFLEXSPIOperation\_Config* FLEXSPI operation: Configure device mode, the TX FIFO size is fixed in LUT.

*kFLEXSPIOperation\_Write* FLEXSPI operation: Write, only TX buffer is effective.

*kFLEXSPIOperation\_Read* FLEXSPI operation: Read, only Rx Buffer is effective.

## 6.5 Function Documentation

### 6.5.1 void ROM\_API\_Init( void )

Get the bootloader api entry address.

### 6.5.2 void ROM\_RunBootloader( void \* arg )

Parameters

---

|            |                                                                                                                         |
|------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>arg</i> | A pointer to the storage for the bootloader param. refer to System Boot Chapter in device reference manual for details. |
|------------|-------------------------------------------------------------------------------------------------------------------------|

### 6.5.3 status\_t ROM\_FLEXSPI\_NorFlash\_GetConfig ( uint32\_t *instance*, flexspi\_nor\_config\_t \* *config*, serial\_nor\_config\_option\_t \* *option* )

Parameters

|                 |                                                                   |
|-----------------|-------------------------------------------------------------------|
| <i>instance</i> | storage the instance of FLEXSPI.                                  |
| <i>config</i>   | A pointer to the storage for the driver runtime state.            |
| <i>option</i>   | A pointer to the storage Serial NOR Configuration Option Context. |

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>          | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |
| <i>kStatus_ROM_FLEXSPI_DeviceTimeout</i>            | the device timeout              |

### 6.5.4 status\_t ROM\_FLEXSPI\_NorFlash\_Init ( uint32\_t *instance*, flexspi\_nor\_config\_t \* *config* )

This function checks and initializes the FLEXSPI module for the other FLEXSPI APIs.

Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>instance</i> | storage the instance of FLEXSPI.                       |
| <i>config</i>   | A pointer to the storage for the driver runtime state. |

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>          | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |
| <i>kStatus_ROM_FLEXSPI_DeviceTimeout</i>            | the device timeout              |

### 6.5.5 status\_t ROM\_FLEXSPI\_NorFlash\_ProgramPage ( *uint32\_t instance*, *flexspi\_nor\_config\_t \* config*, *uint32\_t dst\_addr*, *const uint32\_t \* src* )

This function programs the NOR flash memory with the dest address for a given flash area as determined by the dst address and the length.

Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>instance</i> | storage the instance of FLEXSPI.                        |
| <i>config</i>   | A pointer to the storage for the driver runtime state.  |
| <i>dst_addr</i> | A pointer to the desired flash memory to be programmed. |

Note

It is recommended that use page aligned access; If the dst\_addr is not aligned to page, the driver automatically aligns address down with the page address.

Parameters

|            |                                                                                     |
|------------|-------------------------------------------------------------------------------------|
| <i>src</i> | A pointer to the source buffer of data that is to be programmed into the NOR flash. |
|------------|-------------------------------------------------------------------------------------|

Return values

|                        |                                |
|------------------------|--------------------------------|
| <i>kStatus_Success</i> | Api was executed successfully. |
|------------------------|--------------------------------|

|                                                      |                                 |
|------------------------------------------------------|---------------------------------|
| <i>kStatus_InvalidArgument</i>                       | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI-InvalidSequence</i>           | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI-SequenceExecution-Timeout</i> | Sequence Execution timeout.     |
| <i>kStatus_ROM_FLEXSPI-DeviceTimeout</i>             | the device timeout              |

### 6.5.6 status\_t ROM\_FLEXSPI\_NorFlash\_Read ( *uint32\_t instance*, *flexspi\_nor\_config\_t \* config*, *uint32\_t \* dst*, *uint32\_t start*, *uint32\_t lengthInBytes* )

This function read the NOR flash memory with the start address for a given flash area as determined by the dst address and the length.

#### Parameters

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| <i>instance</i> | storage the instance of FLEXSPI.                                            |
| <i>config</i>   | A pointer to the storage for the driver runtime state.                      |
| <i>dst</i>      | A pointer to the dest buffer of data that is to be read from the NOR flash. |

#### Note

It is recommended that use page aligned access; If the dstAddr is not aligned to page, the driver automatically aligns address down with the page address.

#### Parameters

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| <i>start</i>         | The start address of the desired NOR flash memory to be read. |
| <i>lengthInBytes</i> | The length, given in bytes to be read.                        |

#### Return values

|                        |                                |
|------------------------|--------------------------------|
| <i>kStatus_Success</i> | Api was executed successfully. |
|------------------------|--------------------------------|

|                                                      |                                 |
|------------------------------------------------------|---------------------------------|
| <i>kStatus_InvalidArgument</i>                       | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI-InvalidSequence</i>           | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI-SequenceExecution-Timeout</i> | Sequence Execution timeout.     |
| <i>kStatus_ROM_FLEXSPI-DeviceTimeout</i>             | the device timeout              |

### 6.5.7 **status\_t ROM\_FLEXSPI\_NorFlash\_Erase ( uint32\_t *instance*, flexspi\_nor\_config\_t \* *config*, uint32\_t *start*, uint32\_t *length* )**

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>instance</i> | storage the index of FLEXSPI.                                   |
| <i>config</i>   | A pointer to the storage for the driver runtime state.          |
| <i>start</i>    | The start address of the desired NOR flash memory to be erased. |

Note

It is recommended that use sector-aligned access nor device; If dstAddr is not aligned with the sector, the driver automatically aligns address down with the sector address.

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>length</i> | The length, given in bytes to be erased. |
|---------------|------------------------------------------|

Note

It is recommended that use sector-aligned access nor device; If length is not aligned with the sector, the driver automatically aligns up with the sector.

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>          | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |
| <i>kStatus_ROM_FLEXSPI_DeviceTimeout</i>            | the device timeout              |

### 6.5.8 status\_t ROM\_FLEXSPI\_NorFlash\_EraseSector ( *uint32\_t instance*, *flexspi\_nor\_config\_t \* config*, *uint32\_t start* )

This function erases one of NOR flash sectors based on the desired address.

Parameters

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>instance</i> | storage the index of FLEXSPI.                                   |
| <i>config</i>   | A pointer to the storage for the driver runtime state.          |
| <i>start</i>    | The start address of the desired NOR flash memory to be erased. |

Note

It is recommended that use sector-aligned access nor device; If dstAddr is not aligned with the sector, the driver automatically aligns address down with the sector address.

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>          | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |

|                                          |                    |
|------------------------------------------|--------------------|
| <i>kStatus_ROM_FLEXSPI_DeviceTimeout</i> | the device timeout |
|------------------------------------------|--------------------|

### 6.5.9 status\_t ROM\_FLEXSPI\_NorFlash\_EraseBlock ( uint32\_t *instance*, flexspi\_nor\_config\_t \* *config*, uint32\_t *start* )

This function erases one block of NOR flash based on the desired address.

Parameters

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>instance</i> | storage the index of FLEXSPI.                                   |
| <i>config</i>   | A pointer to the storage for the driver runtime state.          |
| <i>start</i>    | The start address of the desired NOR flash memory to be erased. |

Note

It is recommended that use block-aligned access nor device; If dstAddr is not aligned with the block, the driver automatically aligns address down with the block address.

Return values

|                                                      |                                 |
|------------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                               | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                       | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>           | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecution-Timeout</i> | Sequence Execution timeout.     |
| <i>kStatus_ROM_FLEXSPI_DeviceTimeout</i>             | the device timeout              |

### 6.5.10 status\_t ROM\_FLEXSPI\_NorFlash\_EraseAll ( uint32\_t *instance*, flexspi\_nor\_config\_t \* *config* )

Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>instance</i> | storage the instance of FLEXSPI.                       |
| <i>config</i>   | A pointer to the storage for the driver runtime state. |

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>          | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |
| <i>kStatus_ROM_FLEXSPI_DeviceTimeout</i>            | the device timeout              |

### 6.5.11 **status\_t ROM\_FLEXSPI\_NorFlash\_CommandXfer ( uint32\_t *instance*, flexspi\_xfer\_t \* *xfer* )**

This function is used to perform the command write sequence to the NOR device.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | storage the index of FLEXSPI.                      |
| <i>xfer</i>     | A pointer to the storage FLEXSPI Transfer Context. |

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>          | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |

6.5.12 **status\_t ROM\_FLEXSPI\_NorFlash\_UpdateLut ( uint32\_t *instance*, uint32\_t *seqIndex*, const uint32\_t \* *lutBase*, uint32\_t *seqNumber* )**

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>instance</i>  | storage the index of FLEXSPI.                         |
| <i>seqIndex</i>  | storage the sequence Id.                              |
| <i>lutBase</i>   | A pointer to the look-up-table for command sequences. |
| <i>seqNumber</i> | storage sequence number.                              |

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i>          | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |

### 6.5.13 status\_t ROM\_FLEXSPI\_NorFlash\_WaitBusy ( uint32\_t *instance*, flexspi\_nor\_config\_t \* *config*, bool *isParallelMode*, uint32\_t *address* )

Parameters

|                       |                                                                           |
|-----------------------|---------------------------------------------------------------------------|
| <i>instance</i>       | Indicates the index of FLEXSPI.                                           |
| <i>config</i>         | A pointer to the storage for the driver runtime state                     |
| <i>isParallelMode</i> | Indicates whether NOR flash is in parallel mode.                          |
| <i>address</i>        | Indicates the operation(erase/program/read) address for serial NOR flash. |

Return values

|                                                     |                                 |
|-----------------------------------------------------|---------------------------------|
| <i>kStatus_Success</i>                              | Api was executed successfully.  |
| <i>kStatus_InvalidArgument</i>                      | A invalid argument is provided. |
| <i>kStatus_ROM_FLEXSPI_SequenceExecutionTimeout</i> | Sequence Execution timeout.     |

|                                            |                                 |
|--------------------------------------------|---------------------------------|
| <i>kStatus_ROM_FLEXSPI_InvalidSequence</i> | A invalid Sequence is provided. |
| <i>kStatus_ROM_FLEXSPI_DeviceTimeout</i>   | Device timeout.                 |

### 6.5.14 void ROM\_FLEXSPI\_NorFlash\_ClearCache ( *uint32\_t instance* )

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>instance</i> | storage the index of FLEXSPI. |
|-----------------|-------------------------------|

# Chapter 7

## IOMUXC: IOMUX Controller

### 7.1 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, 0))  
*IOMUXC driver version 2.0.0.*

### Pin function ID

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

- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_SEMC\\_DATA00](#) 0x42A10010U, 0x0U, 0, 0, 0x42-A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_XBAR1\\_XBAR\\_INOUT04](#) 0x42A10010U, 0x1U, 0, 0, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_SINC3\\_MOD\\_CLK0](#) 0x42A10010U, 0x2U, 0, 0, 0x42-A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_LPUART3\\_CTS\\_B](#) 0x42A10010U, 0x3U, 0x42A1067-CU, 0x0U, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_NETC\\_PINMUX\\_ETH3\\_TXD03](#) 0x42A10010U, 0x4-U, 0, 0, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_GPIO2\\_IO00](#) 0x42A10010U, 0x5U, 0, 0, 0x42A10258-U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_KPP\\_ROW03](#) 0x42A10010U, 0x6U, 0x42A105D4U, 0x0U, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_FLEXIO1\\_FLEXIO00](#) 0x42A10010U, 0x8U, 0, 0, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_NETC\\_PINMUX\\_ETH4\\_TXD03](#) 0x42A10010U, 0x9-U, 0, 0, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_ECAT\\_TX\\_DATA3\\_0](#) 0x42A10010U, 0xAU, 0, 0, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_00\\_AHB\\_SRAMC\\_DATA00](#) 0x42A10010U, 0xCU, 0, 0, 0x42A10258U
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_01\\_ECAT\\_TX\\_DATA2\\_0](#) 0x42A10014U, 0xAU, 0, 0, 0x42A1025CU
- #define [IOMUXC\\_GPIO\\_EMC\\_B1\\_01\\_AHB\\_SRAMC\\_DATA01](#) 0x42A10014U, 0xCU, 0, 0, 0x42A1025CU

- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_SEMC\_DATA01** 0x42A10014U, 0x0U, 0, 0, 0x42-A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_XBAR1\_XBAR\_INOUT05** 0x42A10014U, 0x1U, 0, 0, 0x42A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_SINC3\_MOD\_CLK1** 0x42A10014U, 0x2U, 0, 0, 0x42-A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_LPUART3\_RTS\_B** 0x42A10014U, 0x3U, 0, 0, 0x42-A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_NETC\_PINMUX\_ETH3\_TXD02** 0x42A10014U, 0x4-U, 0, 0, 0x42A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_GPIO2\_IO01** 0x42A10014U, 0x5U, 0, 0, 0x42A1025-CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_KPP\_COL03** 0x42A10014U, 0x6U, 0x42A105B4U, 0x0U, 0x42A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_FLEXIO1\_FLEXIO01** 0x42A10014U, 0x8U, 0, 0, 0x42A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_01\_NETC\_PINMUX\_ETH4\_RXD02** 0x42A10014U, 0x9-U, 0, 0, 0x42A1025CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_SEMC\_DATA02** 0x42A10018U, 0x0U, 0, 0, 0x42-A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_XBAR1\_XBAR\_INOUT06** 0x42A10018U, 0x1U, 0, 0, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_SINC3\_MOD\_CLK2** 0x42A10018U, 0x2U, 0, 0, 0x42-A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_LPUART3\_RX** 0x42A10018U, 0x3U, 0x42A10680U, 0x0U, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_NETC\_PINMUX\_ETH3\_RX\_CLK** 0x42A10018U, 0x4U, 0x42A10818U, 0x0U, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_GPIO2\_IO02** 0x42A10018U, 0x5U, 0, 0, 0x42A10260-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_KPP\_ROW02** 0x42A10018U, 0x6U, 0x42A105D0U, 0x0U, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_FLEXIO1\_FLEXIO02** 0x42A10018U, 0x8U, 0, 0, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_NETC\_PINMUX\_ETH4\_RX\_CLK** 0x42A10018U, 0x9U, 0x42A10838U, 0x0U, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_ECAT\_RX\_CLK\_0** 0x42A10018U, 0xAU, 0x42A104-ACU, 0x0U, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_02\_AHB\_SRAMC\_DATA02** 0x42A10018U, 0xCU, 0, 0, 0x42A10260U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_SEMC\_DATA03** 0x42A1001CU, 0x0U, 0, 0, 0x42-A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_XBAR1\_XBAR\_INOUT07** 0x42A1001CU, 0x1U, 0, 0, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_SINC3\_EMCLK00** 0x42A1001CU, 0x2U, 0, 0, 0x42-A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_LPUART3\_TX** 0x42A1001CU, 0x3U, 0x42A10684U, 0x0U, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_NETC\_PINMUX\_ETH3\_RXD03** 0x42A1001CU, 0x4-

- U, 0x42A10830U, 0x0U, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_GPIO2\_IO03** 0x42A1001CU, 0x5U, 0, 0, 0x42A10264-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_KPP\_COL02** 0x42A1001CU, 0x6U, 0x42A105B0U, 0x0U, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_FLEXIO1\_FLEXIO03** 0x42A1001CU, 0x8U, 0, 0, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_NETC\_PINMUX\_ETH4\_RXD03** 0x42A1001CU, 0x9-U, 0x42A10850U, 0x0U, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_ECAT\_RX\_DATA3\_0** 0x42A1001CU, 0xAU, 0x42-A104CCU, 0x0U, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_03\_AHB\_SRAMC\_DATA03** 0x42A1001CU, 0xCU, 0, 0, 0x42A10264U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_ECAT\_RX\_DATA2\_0** 0x42A10020U, 0xAU, 0x42-A104C4U, 0x0U, 0x42A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_AHB\_SRAMC\_DATA04** 0x42A10020U, 0xCU, 0, 0, 0x42A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_SEMC\_DATA04** 0x42A10020U, 0x0U, 0, 0, 0x42-A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_XBAR1\_XBAR\_INOUT08** 0x42A10020U, 0x1U, 0, 0, 0x42A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_SINC3\_EMBIT00** 0x42A10020U, 0x2U, 0, 0, 0x42-A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_LPUART3\_DSR\_B** 0x42A10020U, 0x3U, 0, 0, 0x42-A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_NETC\_PINMUX\_ETH3\_RXD02** 0x42A10020U, 0x4-U, 0x42A1082CU, 0x0U, 0x42A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_GPIO2\_IO04** 0x42A10020U, 0x5U, 0, 0, 0x42A10268-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_KPP\_ROW01** 0x42A10020U, 0x6U, 0x42A105CCU, 0x0U, 0x42A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_FLEXIO1\_FLEXIO04** 0x42A10020U, 0x8U, 0, 0, 0x42A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_04\_NETC\_PINMUX\_ETH4\_RXD02** 0x42A10020U, 0x9-U, 0x42A1084CU, 0x0U, 0x42A10268U
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_SEMC\_DATA05** 0x42A10024U, 0x0U, 0, 0, 0x42-A1026CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_XBAR1\_XBAR\_INOUT09** 0x42A10024U, 0x1U, 0, 0, 0x42A1026CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_SINC3\_EMCLK01** 0x42A10024U, 0x2U, 0, 0, 0x42-A1026CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_LPUART3\_DCD\_B** 0x42A10024U, 0x3U, 0, 0, 0x42-A1026CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_NETC\_PINMUX\_ETH3\_TXD00** 0x42A10024U, 0x4-U, 0, 0, 0x42A1026CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_GPIO2\_IO05** 0x42A10024U, 0x5U, 0, 0, 0x42A1026-CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_KPP\_ROW07** 0x42A10024U, 0x6U, 0x42A105E4U, 0x0U, 0x42A1026CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_FLEXIO1\_FLEXIO05** 0x42A10024U, 0x8U, 0, 0,

- 0x42A1026CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_05\_NETC\_PINMUX\_ETH4\_TXD00** 0x42A10024U, 0x9-U, 0, 0, 0x42A1026CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_05\_ECAT\_TX\_DATA0\_0** 0x42A10024U, 0xAU, 0, 0, 0x42A1026CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_05\_AHB\_SRAMC\_DATA05** 0x42A10024U, 0xCU, 0, 0, 0x42A1026CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_SEMC\_DATA06** 0x42A10028U, 0x0U, 0, 0, 0x42-A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_FLEXPWM4\_PWMB03** 0x42A10028U, 0x1U, 0, 0, 0x42A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_SINC3\_EMBIT01** 0x42A10028U, 0x2U, 0, 0, 0x42-A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_LPUART3\_RL\_B** 0x42A10028U, 0x3U, 0, 0, 0x42-A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_NETC\_PINMUX\_ETH3\_TXD01** 0x42A10028U, 0x4-U, 0, 0, 0x42A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_GPIO2\_IO06** 0x42A10028U, 0x5U, 0, 0, 0x42A10270-U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_KPP\_COL07** 0x42A10028U, 0x6U, 0x42A105C4U, 0x0U, 0x42A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_FLEXIO1\_FLEXIO06** 0x42A10028U, 0x8U, 0, 0, 0x42A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_NETC\_PINMUX\_ETH4\_TXD01** 0x42A10028U, 0x9-U, 0, 0, 0x42A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_ECAT\_TX\_DATA1\_0** 0x42A10028U, 0xAU, 0, 0, 0x42A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_06\_AHB\_SRAMC\_DATA06** 0x42A10028U, 0xCU, 0, 0, 0x42A10270U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_ECAT\_TX\_EN\_0** 0x42A1002CU, 0xAU, 0, 0, 0x42-A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_AHB\_SRAMC\_DATA07** 0x42A1002CU, 0xCU, 0, 0, 0x42A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_SEMC\_DATA07** 0x42A1002CU, 0x0U, 0, 0, 0x42-A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_FLEXPWM4\_PWMA03** 0x42A1002CU, 0x1U, 0, 0, 0x42A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_SINC3\_EMCLK02** 0x42A1002CU, 0x2U, 0, 0, 0x42-A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_LPUART3\_DTR\_B** 0x42A1002CU, 0x3U, 0, 0, 0x42-A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_NETC\_PINMUX\_ETH3\_TX\_EN** 0x42A1002CU, 0x4U, 0, 0, 0x42A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_GPIO2\_IO07** 0x42A1002CU, 0x5U, 0, 0, 0x42A10274-U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_KPP\_ROW06** 0x42A1002CU, 0x6U, 0x42A105E0U, 0x0U, 0x42A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_FLEXIO1\_FLEXIO07** 0x42A1002CU, 0x8U, 0, 0, 0x42A10274U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_07\_NETC\_PINMUX\_ETH4\_TX\_EN** 0x42A1002CU,

- 0x9U, 0, 0, 0x42A10274U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_SEMC\_DM00** 0x42A10030U, 0x0U, 0, 0, 0x42-A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_FLEXPWM2\_PWMB03** 0x42A10030U, 0x1U, 0, 0, 0x42-A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_SINC3\_EMBIT02** 0x42A10030U, 0x2U, 0, 0, 0x42-A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_LPUART4\_DSR\_B** 0x42A10030U, 0x3U, 0, 0, 0x42-A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_NETC\_PINMUX\_ETH3\_TX\_CLK** 0x42A10030U, 0x4U, 0x42A10834U, 0x0U, 0x42A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_GPIO2\_IO08** 0x42A10030U, 0x5U, 0, 0, 0x42A10278-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_KPP\_COL06** 0x42A10030U, 0x6U, 0x42A105C0U, 0x0U, 0x42A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_FLEXIO1\_FLEXIO08** 0x42A10030U, 0x8U, 0, 0, 0x42A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_NETC\_PINMUX\_ETH4\_TX\_CLK** 0x42A10030U, 0x9U, 0x42A10854U, 0x0U, 0x42A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_ECAT\_TX\_CLK\_0** 0x42A10030U, 0xAU, 0x42A104-E4U, 0x0U, 0x42A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_08\_AHB\_SRAMC\_LBB** 0x42A10030U, 0xCU, 0, 0, 0x42-A10278U
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_SEMC\_ADDR00** 0x42A10034U, 0x0U, 0, 0, 0x42-A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_FLEXPWM2\_PWMA03** 0x42A10034U, 0x1U, 0, 0, 0x42-A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_SINC3\_EMCLK03** 0x42A10034U, 0x2U, 0, 0, 0x42-A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_LPUART4\_DCD\_B** 0x42A10034U, 0x3U, 0, 0, 0x42-A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_NETC\_PINMUX\_ETH3\_RXD00** 0x42A10034U, 0x4U, 0x42A10824U, 0x0U, 0x42A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_GPIO2\_IO09** 0x42A10034U, 0x5U, 0, 0, 0x42A1027-CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_KPP\_ROW05** 0x42A10034U, 0x6U, 0x42A105DCU, 0x0U, 0x42A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_FLEXIO1\_FLEXIO09** 0x42A10034U, 0x8U, 0, 0, 0x42A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_NETC\_PINMUX\_ETH4\_RXD00** 0x42A10034U, 0x9U, 0x42A10844U, 0x0U, 0x42A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_ECAT\_RX\_DATA0\_0** 0x42A10034U, 0xAU, 0x42-A104B4U, 0x0U, 0x42A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_09\_AHB\_SRAMC\_ADDR00** 0x42A10034U, 0xCU, 0, 0, 0x42A1027CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_SEMC\_ADDR01** 0x42A10038U, 0x0U, 0, 0, 0x42-A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_FLEXPWM3\_PWMB03** 0x42A10038U, 0x1U, 0x42-A10540U, 0x0U, 0x42A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_SINC3\_EMBIT03** 0x42A10038U, 0x2U, 0, 0, 0x42-

A10280U

- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_LPUART4\_RI\_B** 0x42A10038U, 0x3U, 0, 0, 0x42-A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_NETC\_PINMUX\_ETH3\_RXD01** 0x42A10038U, 0x4-U, 0x42A10828U, 0x0U, 0x42A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_GPIO2\_IO10** 0x42A10038U, 0x5U, 0, 0, 0x42A10280-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_KPP\_COL05** 0x42A10038U, 0x6U, 0x42A105BCU, 0x0U, 0x42A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_FLEXIO1\_FLEXIO10** 0x42A10038U, 0x8U, 0, 0, 0x42A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_NETC\_PINMUX\_ETH4\_RXD01** 0x42A10038U, 0x9-U, 0x42A10848U, 0x0U, 0x42A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_ECAT\_RX\_DATA1\_0** 0x42A10038U, 0xAU, 0x42-A104BCU, 0x0U, 0x42A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_10\_AHB\_SRAMC\_ADDR01** 0x42A10038U, 0xCU, 0, 0, 0x42A10280U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_ECAT\_RX\_DV\_0** 0x42A1003CU, 0xAU, 0x42A104-D4U, 0x0U, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_AHB\_SRAMC\_ADDR02** 0x42A1003CU, 0xCU, 0, 0, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_SEMC\_ADDR02** 0x42A1003CU, 0x0U, 0, 0, 0x42-A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_FLEXPWM3\_PWMA03** 0x42A1003CU, 0x1U, 0x42-A10530U, 0x0U, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_SINC\_FILTER\_GLUE3\_BREAK** 0x42A1003CU, 0x2U, 0, 0, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_LPUART4\_DTR\_B** 0x42A1003CU, 0x3U, 0, 0, 0x42-A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_NETC\_PINMUX\_ETH3\_RX\_DV** 0x42A1003CU, 0x4U, 0x42A1081CU, 0x0U, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_GPIO2\_IO11** 0x42A1003CU, 0x5U, 0, 0, 0x42A10284-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_KPP\_ROW04** 0x42A1003CU, 0x6U, 0x42A105D8U, 0x0U, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_FLEXIO1\_FLEXIO11** 0x42A1003CU, 0x8U, 0, 0, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_11\_NETC\_PINMUX\_ETH4\_RX\_DV** 0x42A1003CU, 0x9U, 0x42A1083CU, 0x0U, 0x42A10284U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_SEMC\_ADDR03** 0x42A10040U, 0x0U, 0, 0, 0x42-A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_FLEXPWM4\_PWMA00** 0x42A10040U, 0x1U, 0, 0, 0x42A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_LPUART4\_TX** 0x42A10040U, 0x2U, 0, 0, 0x42-A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_NETC\_PINMUX\_ETH3\_RX\_ER** 0x42A10040U, 0x4-U, 0x42A10820U, 0x0U, 0x42A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_GPIO2\_IO12** 0x42A10040U, 0x5U, 0, 0, 0x42A10288-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_KPP\_COL04** 0x42A10040U, 0x6U, 0x42A105B8U,

- 0x0U, 0x42A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_FLEXIO1\_FLEXIO12** 0x42A10040U, 0x8U, 0, 0, 0x42A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_NETC\_PINMUX\_ETH4\_RX\_ER** 0x42A10040U, 0x9-U, 0x42A10840U, 0x0U, 0x42A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_ECAT\_PT0\_RX\_ER** 0x42A10040U, 0xAU, 0x42-A104DCU, 0x0U, 0x42A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_12\_AHB\_SRAMC\_ADDR03** 0x42A10040U, 0xCU, 0, 0, 0x42A10288U
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_SEMC\_ADDR04** 0x42A10044U, 0x0U, 0, 0, 0x42-A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_FLEXPWM4\_PWMB00** 0x42A10044U, 0x1U, 0, 0, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_LPUART4\_RX** 0x42A10044U, 0x2U, 0, 0, 0x42-A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_NETC\_PINMUX\_ETH2\_RX\_DV** 0x42A10044U, 0x3U, 0x42A107FCU, 0x0U, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_NETC\_PINMUX\_ETH3\_TX\_ER** 0x42A10044U, 0x4-U, 0, 0, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_GPIO2\_IO13** 0x42A10044U, 0x5U, 0, 0, 0x42A1028-CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_KPP\_COL01** 0x42A10044U, 0x6U, 0x42A105ACU, 0x0U, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_FLEXIO1\_FLEXIO13** 0x42A10044U, 0x8U, 0, 0, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_NETC\_PINMUX\_ETH4\_TX\_ER** 0x42A10044U, 0x9-U, 0, 0, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_QTIMER1\_TIMER1** 0x42A10044U, 0xAU, 0x42-A1085CU, 0x0U, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_13\_AHB\_SRAMC\_ADDR04** 0x42A10044U, 0xCU, 0, 0, 0x42A1028CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_LPUART4\_CTS\_B** 0x42A10048U, 0xAU, 0x42-A10688U, 0x0U, 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_AHB\_SRAMC\_ADDR05** 0x42A10048U, 0xCU, 0, 0, 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_SEMC\_ADDR05** 0x42A10048U, 0x0U, 0, 0, 0x42-A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_FLEXPWM4\_PWMA01** 0x42A10048U, 0x1U, 0, 0, 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_LPUART5\_TX** 0x42A10048U, 0x2U, 0x42A106A0U, 0x0U, 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_NETC\_PINMUX\_ETH2\_TX\_EN** 0x42A10048U, 0x3-U, 0, 0, 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_NETC\_ETH3 CRS** 0x42A10048U, 0x4U, 0x42A107-B0U, 0x0U, 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_GPIO2\_IO14** 0x42A10048U, 0x5U, 0, 0, 0x42A10290-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_KPP\_ROW00** 0x42A10048U, 0x6U, 0x42A105C8U, 0x0U, 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_FLEXIO1\_FLEXIO14** 0x42A10048U, 0x8U, 0, 0,

- 0x42A10290U
- #define **IOMUXC\_GPIO\_EMC\_B1\_14\_NETC\_ETH4\_CRS** 0x42A10048U, 0x9U, 0x42A107-C0U, 0x0U, 0x42A10290U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_SEMC\_ADDR06** 0x42A1004CU, 0x0U, 0, 0, 0x42-A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_FLEXPWM4\_PWMB01** 0x42A1004CU, 0x1U, 0, 0, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_LPUART5\_RX** 0x42A1004CU, 0x2U, 0x42A1069CU, 0x0U, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_NETC\_PINMUX\_ETH2\_TX\_CLK** 0x42A1004CU, 0x3U, 0x42A10814U, 0x0U, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_NETC\_ETH3\_COL** 0x42A1004CU, 0x4U, 0x42A107-ACU, 0x0U, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_GPIO2\_IO15** 0x42A1004CU, 0x5U, 0, 0, 0x42A10294-U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_KPP\_COL00** 0x42A1004CU, 0x6U, 0x42A105A8U, 0x0U, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_FLEXIO1\_FLEXIO15** 0x42A1004CU, 0x8U, 0, 0, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_NETC\_ETH4\_COL** 0x42A1004CU, 0x9U, 0x42A107-BCU, 0x0U, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_LPUART4\_RTS\_B** 0x42A1004CU, 0xAU, 0, 0, 0x42-A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_15\_AHB\_SRAMC\_ADDR06** 0x42A1004CU, 0xCU, 0, 0, 0x42A10294U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_SEMC\_ADDR07** 0x42A10050U, 0x0U, 0, 0, 0x42-A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_FLEXPWM4\_PWMB02** 0x42A10050U, 0x1U, 0, 0, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_LPUART9\_TX** 0x42A10050U, 0x2U, 0x42A106CCU, 0x0U, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_NETC\_PINMUX\_ETH2\_RXD00** 0x42A10050U, 0x3-U, 0x42A10804U, 0x0U, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_NETC\_ETH3\_SLV\_MDC** 0x42A10050U, 0x4U, 0x42A107B4U, 0x0U, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_GPIO2\_IO16** 0x42A10050U, 0x5U, 0, 0, 0x42A10298-U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_NETC\_ETH4\_SLV\_MDC** 0x42A10050U, 0x6U, 0x42A107C4U, 0x0U, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_FLEXIO1\_FLEXIO16** 0x42A10050U, 0x8U, 0, 0, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_NETC\_ETH2\_SLV\_MDC** 0x42A10050U, 0x9U, 0x42A107A4U, 0x0U, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_LPSPi6\_PCS2** 0x42A10050U, 0xAU, 0x42A10658U, 0x0U, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_16\_AHB\_SRAMC\_ADDR07** 0x42A10050U, 0xCU, 0, 0, 0x42A10298U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_LPSPi6\_PCS1** 0x42A10054U, 0xAU, 0x42A10654U, 0x0U, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_AHB\_SRAMC\_ADDR08** 0x42A10054U, 0xCU, 0, 0,

- 0x42A1029CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_17\_SEMC\_ADDR08** 0x42A10054U, 0x0U, 0, 0, 0x42-A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_FLEXPWM4\_PWMA02** 0x42A10054U, 0x1U, 0, 0, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_LPUART9\_RX** 0x42A10054U, 0x2U, 0x42A106C8U, 0x0U, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_NETC\_PINMUX\_ETH2\_RXD01** 0x42A10054U, 0x3-U, 0x42A10808U, 0x0U, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_NETC\_ETH3\_SLV\_MDIO** 0x42A10054U, 0x4U, 0x42A107B8U, 0x0U, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_GPIO2\_IO17** 0x42A10054U, 0x5U, 0, 0, 0x42A1029-CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_NETC\_ETH4\_SLV\_MDIO** 0x42A10054U, 0x6U, 0x42A107C8U, 0x0U, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_FLEXIO1\_FLEXIO17** 0x42A10054U, 0x8U, 0, 0, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_17\_NETC\_ETH2\_SLV\_MDIO** 0x42A10054U, 0x9U, 0x42A107A8U, 0x0U, 0x42A1029CU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_SEMC\_ADDR09** 0x42A10058U, 0x0U, 0, 0, 0x42-A102A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_FLEXPWM2\_PWMA00** 0x42A10058U, 0x1U, 0x42-A1050CU, 0x0U, 0x42A102A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_QTIMER1\_TIMER0** 0x42A10058U, 0x2U, 0x42-A10858U, 0x0U, 0x42A102A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_LPSPi6\_SCK** 0x42A10058U, 0x3U, 0x42A10660U, 0x0U, 0x42A102A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_NETC\_ETH2\_CRS** 0x42A10058U, 0x4U, 0x42A107-A0U, 0x0U, 0x42A102A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_GPIO2\_IO18** 0x42A10058U, 0x5U, 0, 0, 0x42A102A0-U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_FLEXIO1\_FLEXIO18** 0x42A10058U, 0x8U, 0, 0, 0x42A102A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_NETC\_EMDC** 0x42A10058U, 0xAU, 0, 0, 0x42A102-A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_18\_AHB\_SRAMC\_ADDR09** 0x42A10058U, 0xCU, 0, 0, 0x42A102A0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_SEMC\_ADDR11** 0x42A1005CU, 0x0U, 0, 0, 0x42-A102A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_FLEXPWM2\_PWMB00** 0x42A1005CU, 0x1U, 0x42-A10518U, 0x0U, 0x42A102A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_QTIMER2\_TIMER0** 0x42A1005CU, 0x2U, 0x42-A10864U, 0x0U, 0x42A102A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_LPSPi6\_SDI** 0x42A1005CU, 0x3U, 0x42A10664U, 0x0U, 0x42A102A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_NETC\_ETH2\_COL** 0x42A1005CU, 0x4U, 0x42-A1079CU, 0x0U, 0x42A102A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_GPIO2\_IO19** 0x42A1005CU, 0x5U, 0, 0, 0x42A102-A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_FLEXIO1\_FLEXIO19** 0x42A1005CU, 0x8U, 0, 0,

- 0x42A102A4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_19\_NETC\_EMDIO** 0x42A1005CU, 0xAU, 0x42A10798U, 0x0U, 0x42A102A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_19\_AHB\_SRAMC\_ADDR11** 0x42A1005CU, 0xCU, 0, 0, 0x42A102A4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_SEMC\_ADDR12** 0x42A10060U, 0x0U, 0, 0, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_FLEXPWM2\_PWMA01** 0x42A10060U, 0x1U, 0x42A10510U, 0x0U, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_QTIMER3\_TIMER0** 0x42A10060U, 0x2U, 0x42A10870U, 0x0U, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_LPSPi6\_SDO** 0x42A10060U, 0x3U, 0x42A10668U, 0x0U, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_NETC\_PINMUX\_ETH2\_TX\_ER** 0x42A10060U, 0x4U, 0, 0, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_GPIO2\_IO20** 0x42A10060U, 0x5U, 0, 0, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_FLEXIO1\_FLEXIO20** 0x42A10060U, 0x8U, 0, 0, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_20\_AHB\_SRAMC\_ADDR12** 0x42A10060U, 0xCU, 0, 0, 0x42A102A8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_FLEXSPI2\_BUS2BIT\_B\_DQS** 0x42A10064U, 0xAU, 0x42A10574U, 0x0U, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_AHB\_SRAMC\_ADDR13** 0x42A10064U, 0xCU, 0, 0, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_SEMC\_BA0** 0x42A10064U, 0x0U, 0, 0, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_FLEXPWM2\_PWMB01** 0x42A10064U, 0x1U, 0x42A1051CU, 0x0U, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_QTIMER4\_TIMER0** 0x42A10064U, 0x2U, 0x42A1087CU, 0x0U, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_LPSPi6\_PCS0** 0x42A10064U, 0x3U, 0x42A10650U, 0x0U, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_NETC\_PINMUX\_ETH2\_RX\_CLK** 0x42A10064U, 0x4U, 0x42A107F8U, 0x0U, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_GPIO2\_IO21** 0x42A10064U, 0x5U, 0, 0, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_FLEXIO1\_FLEXIO21** 0x42A10064U, 0x8U, 0, 0, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_21\_LPUART4\_CTS\_B** 0x42A10064U, 0x9U, 0x42A10688U, 0x1U, 0x42A102ACU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_22\_FLEXSPI2\_BUS2BIT\_B\_DATA03** 0x42A10068U, 0xAU, 0x42A10594U, 0x0U, 0x42A102B0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_22\_AHB\_SRAMC\_ADDR14** 0x42A10068U, 0xCU, 0, 0, 0x42A102B0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_22\_SEMC\_BA1** 0x42A10068U, 0x0U, 0, 0, 0x42A102B0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_22\_FLEXPWM2\_PWMB02** 0x42A10068U, 0x1U, 0x42A10520U, 0x0U, 0x42A102B0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_22\_QTIMER5\_TIMER0** 0x42A10068U, 0x2U, 0x42A10888U, 0x0U, 0x42A102B0U

- #define **IOMUXC\_GPIO\_EMC\_B1\_22\_LPSPi4\_SCK** 0x42A10068U, 0x3U, 0x42A10628U, 0x0U, 0x42A102B0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_22\_NETC\_PINMUX\_ETH2\_RXD02** 0x42A10068U, 0x4U, 0x42A1080CU, 0x0U, 0x42A102B0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_22\_GPIO2\_IO22** 0x42A10068U, 0x5U, 0, 0, 0x42A102B0-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_22\_FLEXIO1\_FLEXIO22** 0x42A10068U, 0x8U, 0, 0, 0x42A102B0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_22\_LPUART4\_RTS\_B** 0x42A10068U, 0x9U, 0, 0, 0x42A102B0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_SEMC\_ADDR10** 0x42A1006CU, 0x0U, 0, 0, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_FLEXPWM2\_PWMA02** 0x42A1006CU, 0x1U, 0x42A10514U, 0x0U, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_QTIMER6\_TIMER0** 0x42A1006CU, 0x2U, 0x42A10894U, 0x0U, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_LPSPi4\_SDI** 0x42A1006CU, 0x3U, 0x42A1062CU, 0x0U, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_NETC\_PINMUX\_ETH2\_RXD03** 0x42A1006CU, 0x4U, 0x42A10810U, 0x0U, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_GPIO2\_IO23** 0x42A1006CU, 0x5U, 0, 0, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_FLEXIO1\_FLEXIO23** 0x42A1006CU, 0x8U, 0, 0, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_FLEXSPI2\_BUS2BIT\_B\_DATA02** 0x42A1006CU, 0xAU, 0x42A10590U, 0x0U, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_23\_AHB\_SRAMC\_ADDR10** 0x42A1006CU, 0xCU, 0, 0, 0x42A102B4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_FLEXSPI2\_BUS2BIT\_B\_DATA01** 0x42A10070U, 0xAU, 0x42A1058CU, 0x0U, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_AHB\_SRAMC\_ADDR15** 0x42A10070U, 0xCU, 0, 0, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_SEMC\_CAS** 0x42A10070U, 0x0U, 0, 0, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_FLEXPWM1\_PWMA00** 0x42A10070U, 0x1U, 0x42A104F4U, 0x0U, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_QTIMER7\_TIMER0** 0x42A10070U, 0x2U, 0x42A108A0U, 0x0U, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_LPSPi4\_SDO** 0x42A10070U, 0x3U, 0x42A10630U, 0x0U, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_NETC\_PINMUX\_ETH2\_TXD03** 0x42A10070U, 0x4U, 0, 0, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_GPIO2\_IO24** 0x42A10070U, 0x5U, 0, 0, 0x42A102B8-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_FLEXIO1\_FLEXIO24** 0x42A10070U, 0x8U, 0, 0, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_24\_NETC\_ETH3\_SLV\_MDC** 0x42A10070U, 0x9U, 0x42A107B4U, 0x1U, 0x42A102B8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_25\_FLEXSPI2\_BUS2BIT\_B\_DATA00** 0x42A10074U, 0xAU, 0x42A10588U, 0x0U, 0x42A102BCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_25\_AHB\_SRAMC\_ADDR16** 0x42A10074U, 0xCU, 0, 0,

- 0x42A102BCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_25\_SEMC\_RAS** 0x42A10074U, 0x0U, 0, 0, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_25\_FLEXPWM1\_PWMB00** 0x42A10074U, 0x1U, 0x42A10500U, 0x0U, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_25\_QTIMER8\_TIMER0** 0x42A10074U, 0x2U, 0x42A108A8U, 0x0U, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_25\_LPSPI4\_PCS0** 0x42A10074U, 0x3U, 0x42A10624U, 0x0U, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_25\_NETC\_PINMUX\_ETH2\_TXD02** 0x42A10074U, 0x4U, 0, 0, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_25\_GPIO2\_IO25** 0x42A10074U, 0x5U, 0, 0, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_25\_FLEXIO1\_FLEXIO25** 0x42A10074U, 0x8U, 0, 0, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_25\_NETC\_ETH3\_SLV\_MDIO** 0x42A10074U, 0x9U, 0x42A107B8U, 0x1U, 0x42A102BCU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_SEMC\_CLK** 0x42A10078U, 0x0U, 0, 0, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_FLEXPWM1\_PWMA01** 0x42A10078U, 0x1U, 0x42A104F8U, 0x0U, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_XBAR1\_XBAR\_INOUT10** 0x42A10078U, 0x2U, 0, 0, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_FLEXSPI2\_BUS2BIT\_A\_SS1\_B** 0x42A10078U, 0x3U, 0, 0, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_NETC\_PINMUX\_ETH2\_TXD01** 0x42A10078U, 0x4U, 0, 0, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_GPIO2\_IO26** 0x42A10078U, 0x5U, 0, 0, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_ECAT\_TX\_DATA1\_1** 0x42A10078U, 0x6U, 0, 0, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_LPSPI6\_SCK** 0x42A10078U, 0xAU, 0x42A10660U, 0x1U, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_26\_AHB\_SRAMC\_WE** 0x42A10078U, 0xCU, 0, 0, 0x42A102C0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_LPSPI6\_SDI** 0x42A1007CU, 0xAU, 0x42A10664U, 0x1U, 0x42A102C4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_AHB\_SRAMC\_OEB** 0x42A1007CU, 0xCU, 0, 0, 0x42A102C4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_SEMC\_CKE** 0x42A1007CU, 0x0U, 0, 0, 0x42A102C4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_FLEXPWM1\_PWMB01** 0x42A1007CU, 0x1U, 0x42A10504U, 0x0U, 0x42A102C4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_XBAR1\_XBAR\_INOUT11** 0x42A1007CU, 0x2U, 0, 0, 0x42A102C4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_FLEXSPI2\_BUS2BIT\_B\_SS1\_B** 0x42A1007CU, 0x3U, 0, 0, 0x42A102C4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_NETC\_PINMUX\_ETH2\_TXD00** 0x42A1007CU, 0x4U, 0, 0, 0x42A102C4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_27\_GPIO2\_IO27** 0x42A1007CU, 0x5U, 0, 0, 0x42A102C4U

C4U

- #define **IOMUXC\_GPIO\_EMC\_B1\_27\_ECAT\_TX\_DATA0\_1** 0x42A1007CU, 0x6U, 0, 0, 0x42A102C4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_27\_LPUART6\_RI\_B** 0x42A1007CU, 0x9U, 0x42A106B0-U, 0x0U, 0x42A102C4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_LPSPi6\_SDO** 0x42A10080U, 0xAU, 0x42A10668U, 0x1U, 0x42A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_AHB\_SRAMC\_ADV** 0x42A10080U, 0xCU, 0, 0, 0x42-A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_SEMC\_WE** 0x42A10080U, 0x0U, 0, 0, 0x42A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_FLEXPWM1\_PWMB02** 0x42A10080U, 0x1U, 0x42-A10508U, 0x0U, 0x42A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_XBAR1\_XBAR\_INOUT12** 0x42A10080U, 0x2U, 0, 0, 0x42A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_FLEXSPI2\_BUS2BIT\_B\_SS0\_B** 0x42A10080U, 0x3-U, 0, 0, 0x42A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_NETC\_PINMUX\_ETH2\_TX\_EN** 0x42A10080U, 0x4-U, 0, 0, 0x42A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_GPIO2\_IO28** 0x42A10080U, 0x5U, 0, 0, 0x42A102C8-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_ECAT\_TX\_EN\_1** 0x42A10080U, 0x6U, 0, 0, 0x42-A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_28\_LPUART6\_DTR\_B** 0x42A10080U, 0x9U, 0, 0, 0x42-A102C8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_SEMC\_CS0** 0x42A10084U, 0x0U, 0, 0, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_FLEXPWM1\_PWMA02** 0x42A10084U, 0x1U, 0x42-A104FCU, 0x0U, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_XBAR1\_XBAR\_INOUT13** 0x42A10084U, 0x2U, 0, 0, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_FLEXSPI2\_BUS2BIT\_B\_DQS** 0x42A10084U, 0x3U, 0x42A10574U, 0x1U, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_NETC\_PINMUX\_ETH2\_TX\_CLK** 0x42A10084U, 0x4U, 0x42A10814U, 0x1U, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_GPIO2\_IO29** 0x42A10084U, 0x5U, 0, 0, 0x42A102C-CU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_ECAT\_TX\_CLK\_1** 0x42A10084U, 0x6U, 0x42A104-E8U, 0x0U, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_LPUART6\_DCD\_B** 0x42A10084U, 0x9U, 0x42A106-A8U, 0x0U, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_LPSPi6\_PCS0** 0x42A10084U, 0xAU, 0x42A10650U, 0x1U, 0x42A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_29\_AHB\_SRAMC\_CS0** 0x42A10084U, 0xCU, 0, 0, 0x42-A102CCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_30\_SEMC\_DATA08** 0x42A10088U, 0x0U, 0, 0, 0x42-A102D0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_30\_FLEXPWM3\_PWMA00** 0x42A10088U, 0x1U, 0x42-A10524U, 0x0U, 0x42A102D0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_30\_XBAR1\_XBAR\_INOUT14** 0x42A10088U, 0x2U, 0x42A10934U, 0x0U, 0x42A102D0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_30\_FLEXSPI2\_BUS2BIT\_B\_DATA03** 0x42A10088U,

- 0x3U, 0x42A10594U, 0x1U, 0x42A102D0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_30\_NETC\_PINMUX\_ETH2\_RXD00** 0x42A10088U, 0x4-U, 0x42A10804U, 0x1U, 0x42A102D0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_30\_GPIO2\_IO30** 0x42A10088U, 0x5U, 0, 0, 0x42A102D0-U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_30\_ECAT\_RX\_DATA0\_1** 0x42A10088U, 0x6U, 0x42-A104B8U, 0x0U, 0x42A102D0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_30\_LPUART6\_DSR\_B** 0x42A10088U, 0x9U, 0x42A106-ACU, 0x0U, 0x42A102D0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_30\_LPSPi6\_PCS1** 0x42A10088U, 0xAU, 0x42A10654U, 0x1U, 0x42A102D0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_30\_AHB\_SRAMC\_DATA08** 0x42A10088U, 0xCU, 0, 0, 0x42A102D0U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_LPSPi6\_PCS2** 0x42A1008CU, 0xAU, 0x42A10658U, 0x1U, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_AHB\_SRAMC\_DATA09** 0x42A1008CU, 0xCU, 0, 0, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_SEMC\_DATA09** 0x42A1008CU, 0x0U, 0, 0, 0x42-A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_FLEXPWM3\_PWMB00** 0x42A1008CU, 0x1U, 0x42-A10534U, 0x0U, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_LPUART6\_TX** 0x42A1008CU, 0x2U, 0x42A106B8U, 0x0U, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_FLEXSPI2\_BUS2BIT\_B\_DATA02** 0x42A1008CU, 0x3U, 0x42A10590U, 0x1U, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_NETC\_PINMUX\_ETH2\_RXD01** 0x42A1008CU, 0x4-U, 0x42A10808U, 0x1U, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_GPIO2\_IO31** 0x42A1008CU, 0x5U, 0, 0, 0x42A102-D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_ECAT\_RX\_DATA1\_1** 0x42A1008CU, 0x6U, 0x42-A104C0U, 0x0U, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_31\_LPSPi5\_SCK** 0x42A1008CU, 0x9U, 0x42A10644U, 0x0U, 0x42A102D4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_LPSPi6\_PCS3** 0x42A10090U, 0xAU, 0x42A1065CU, 0x0U, 0x42A102D8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_AHB\_SRAMC\_DATA10** 0x42A10090U, 0xCU, 0, 0, 0x42A102D8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_SEMC\_DATA10** 0x42A10090U, 0x0U, 0, 0, 0x42-A102D8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_FLEXPWM3\_PWMA01** 0x42A10090U, 0x1U, 0x42-A10528U, 0x0U, 0x42A102D8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_LPUART6\_RX** 0x42A10090U, 0x2U, 0x42A106B4U, 0x0U, 0x42A102D8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_FLEXSPI2\_BUS2BIT\_B\_DATA01** 0x42A10090U, 0x3U, 0x42A1058CU, 0x1U, 0x42A102D8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_NETC\_PINMUX\_ETH2\_RX\_DV** 0x42A10090U, 0x4U, 0x42A107FCU, 0x1U, 0x42A102D8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_GPIO3\_IO00** 0x42A10090U, 0x5U, 0, 0, 0x42A102D8-U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_32\_ECAT\_RX\_DV\_1** 0x42A10090U, 0x6U, 0x42A104D8-

- U, 0x0U, 0x42A102D8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_32\_LPSPi5\_SDO** 0x42A10090U, 0x9U, 0x42A1064CU, 0x0U, 0x42A102D8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_SEMC\_DATA11** 0x42A10094U, 0x0U, 0, 0, 0x42-A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_FLEXPWM3\_PWMB01** 0x42A10094U, 0x1U, 0x42-A10538U, 0x0U, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_LPUART6\_CTS\_B** 0x42A10094U, 0x2U, 0x42A106-A4U, 0x0U, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_FLEXSPI2\_BUS2BIT\_B\_DATA00** 0x42A10094U, 0x3U, 0x42A10588U, 0x1U, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_NETC\_PINMUX\_ETH2\_RX\_ER** 0x42A10094U, 0x4U, 0x42A10800U, 0x0U, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_GPIO3\_IO01** 0x42A10094U, 0x5U, 0, 0, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_ECAT\_RX\_ER\_1** 0x42A10094U, 0x6U, 0x42A104E0-U, 0x0U, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_LPSPi5\_SDI** 0x42A10094U, 0x9U, 0x42A10648U, 0x0U, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_NETC\_PINMUX\_ETH2\_RX\_CLK** 0x42A10094U, 0xAU, 0x42A107F8U, 0x1U, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_33\_AHB\_SRAMC\_DATA11** 0x42A10094U, 0xCU, 0, 0, 0x42A102DCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_LPSPi5\_PCS0** 0x42A10098U, 0xAU, 0x42A10634U, 0x0U, 0x42A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_AHB\_SRAMC\_DATA12** 0x42A10098U, 0xCU, 0, 0, 0x42A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_SEMC\_DATA12** 0x42A10098U, 0x0U, 0, 0, 0x42-A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_FLEXPWM3\_PWMB02** 0x42A10098U, 0x1U, 0x42-A1053CU, 0x0U, 0x42A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_LPUART6\_RTS\_B** 0x42A10098U, 0x2U, 0, 0, 0x42-A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_FLEXSPI2\_BUS2BIT\_B\_SCLK** 0x42A10098U, 0x3U, 0x42A1059CU, 0x0U, 0x42A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_NETC\_PINMUX\_ETH2\_RXD02** 0x42A10098U, 0x4U, 0x42A1080CU, 0x1U, 0x42A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_GPIO3\_IO02** 0x42A10098U, 0x5U, 0, 0, 0x42A102E0-U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_ECAT\_RX\_DATA2\_1** 0x42A10098U, 0x6U, 0x42-A104C8U, 0x0U, 0x42A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_34\_NETC\_PINMUX\_ETH0\_TXD00** 0x42A10098U, 0x9U, 0, 0, 0x42A102E0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_35\_LPSPi5\_PCS1** 0x42A1009CU, 0xAU, 0x42A10638U, 0x0U, 0x42A102E4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_35\_AHB\_SRAMC\_DATA13** 0x42A1009CU, 0xCU, 0, 0, 0x42A102E4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_35\_SEMC\_DATA13** 0x42A1009CU, 0x0U, 0, 0, 0x42-A102E4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_35\_FLEXPWM3\_PWMA02** 0x42A1009CU, 0x1U, 0x42-

- A1052CU, 0x0U, 0x42A102E4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_35\_LPUART5\_TX** 0x42A1009CU, 0x2U, 0x42A106A0U, 0x1U, 0x42A102E4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_35\_FLEXSPI2\_BUS2BIT\_A\_DATA00** 0x42A1009CU, 0x3U, 0x42A10578U, 0x0U, 0x42A102E4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_35\_NETC\_PINMUX\_ETH2\_RXD03** 0x42A1009CU, 0x4U, 0x42A10810U, 0x1U, 0x42A102E4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_35\_GPIO3\_IO03** 0x42A1009CU, 0x5U, 0, 0, 0x42A102E4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_35\_ECAT\_RX\_DATA3\_1** 0x42A1009CU, 0x6U, 0x42A104D0U, 0x0U, 0x42A102E4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_35\_NETC\_PINMUX\_ETH0\_TXD01** 0x42A1009CU, 0x9U, 0, 0, 0x42A102E4U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_SEMC\_DATA14** 0x42A100A0U, 0x0U, 0, 0, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_FLEXPWM1\_PWMA00** 0x42A100A0U, 0x1U, 0x42A104F4U, 0x1U, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_LPUART5\_RX** 0x42A100A0U, 0x2U, 0x42A1069CU, 0x1U, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_FLEXSPI2\_BUS2BIT\_A\_DATA01** 0x42A100A0U, 0x3U, 0x42A1057CU, 0x0U, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_NETC\_PINMUX\_ETH2\_TXD03** 0x42A100A0U, 0x4U, 0, 0, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_GPIO3\_IO04** 0x42A100A0U, 0x5U, 0, 0, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_ECAT\_TX\_DATA3\_1** 0x42A100A0U, 0x6U, 0, 0, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_NETC\_PINMUX\_ETH0\_TX\_EN** 0x42A100A0U, 0x9U, 0, 0, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_36\_AHB\_SRAMC\_DATA14** 0x42A100A0U, 0xCU, 0, 0, 0x42A102E8U
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_AHB\_SRAMC\_DATA15** 0x42A100A4U, 0xCU, 0, 0, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_SEMC\_DATA15** 0x42A100A4U, 0x0U, 0, 0, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_FLEXPWM1\_PWMB00** 0x42A100A4U, 0x1U, 0x42A10500U, 0x1U, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_LPUART5\_CTS\_B** 0x42A100A4U, 0x2U, 0x42A1068CU, 0x0U, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_FLEXSPI2\_BUS2BIT\_A\_DATA02** 0x42A100A4U, 0x3U, 0x42A10580U, 0x0U, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_NETC\_PINMUX\_ETH2\_TXD02** 0x42A100A4U, 0x4U, 0, 0, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_GPIO3\_IO05** 0x42A100A4U, 0x5U, 0, 0, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_ECAT\_TX\_DATA2\_1** 0x42A100A4U, 0x6U, 0, 0, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_37\_NETC\_PINMUX\_ETH0\_TX\_CLK** 0x42A100A4U, 0x9U, 0x42A107F4U, 0x0U, 0x42A102ECU
  - #define **IOMUXC\_GPIO\_EMC\_B1\_38\_AHB\_SRAMC\_UBB** 0x42A100A8U, 0xCU, 0, 0,

- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_SEMC\_DM01** 0x42A100A8U, 0x0U, 0, 0, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_FLEXPWM1\_PWMB03** 0x42A100A8U, 0x1U, 0, 0, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_LPUART5\_RTS\_B** 0x42A100A8U, 0x2U, 0, 0, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_FLEXSPI2\_BUS2BIT\_A\_DATA03** 0x42A100A8U, 0x3U, 0x42A10584U, 0x0U, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_NETC\_PINMUX\_ETH2\_RX\_CLK** 0x42A100A8U, 0x4U, 0x42A107F8U, 0x2U, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_GPIO3\_IO06** 0x42A100A8U, 0x5U, 0, 0, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_ECAT\_RX\_CLK\_1** 0x42A100A8U, 0x6U, 0x42A104B0U, 0x0U, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_38\_NETC\_PINMUX\_ETH0\_RXD00** 0x42A100A8U, 0x9U, 0x42A107E4U, 0x0U, 0x42A102F0U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_SEMC\_DQS** 0x42A100ACU, 0x0U, 0, 0, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_FLEXPWM1\_PWMA03** 0x42A100ACU, 0x1U, 0, 0, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_XBAR1\_XBAR\_INOUT15** 0x42A100ACU, 0x2U, 0x42A10938U, 0x0U, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_FLEXSPI2\_BUS2BIT\_A\_SS0\_B** 0x42A100ACU, 0x3U, 0, 0, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_NETC\_PINMUX\_ETH2\_TX\_ER** 0x42A100ACU, 0x4U, 0, 0, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_GPIO3\_IO07** 0x42A100ACU, 0x5U, 0, 0, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_QTIMER2\_TIMER1** 0x42A100ACU, 0x6U, 0x42A10868U, 0x0U, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_NETC\_PINMUX\_ETH0\_RXD01** 0x42A100ACU, 0x9U, 0x42A107E8U, 0x0U, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_39\_AHB\_SRAMC\_CS1** 0x42A100ACU, 0x1CU, 0, 0, 0x42A102F4U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_SEMC\_RDY** 0x42A100B0U, 0x0U, 0, 0, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_NETC\_EMDC** 0x42A100B0U, 0x1U, 0, 0, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_NETC\_ETH2\_SLV\_MDC** 0x42A100B0U, 0x2U, 0x42A107A4U, 0x1U, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_FLEXSPI2\_BUS2BIT\_A\_DQS** 0x42A100B0U, 0x3U, 0x42A10570U, 0x0U, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_NETC\_ETH2\_CRS** 0x42A100B0U, 0x4U, 0x42A107A0U, 0x1U, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_GPIO3\_IO08** 0x42A100B0U, 0x5U, 0, 0, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_QTIMER3\_TIMER1** 0x42A100B0U, 0x6U, 0x42A10874U, 0x0U, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_NETC\_PINMUX\_ETH0\_RX\_DV** 0x42A100B0U,

- 0x9U, 0x42A107DCU, 0x0U, 0x42A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_40\_AHB\_SRAMC\_CS2** 0x42A100B0U, 0xCU, 0, 0, 0x42-A102F8U
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_AHB\_SRAMC\_CS3** 0x42A100B4U, 0xCU, 0, 0, 0x42-A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_SEMC\_CSX00** 0x42A100B4U, 0x0U, 0, 0, 0x42A102-FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_NETC\_EMDIO** 0x42A100B4U, 0x1U, 0x42A10798U, 0x1U, 0x42A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_NETC\_ETH2\_SLV\_MDIO** 0x42A100B4U, 0x2U, 0x42A107A8U, 0x1U, 0x42A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_FLEXSPI2\_BUS2BIT\_A\_SCLK** 0x42A100B4U, 0x3-U, 0x42A10598U, 0x0U, 0x42A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_NETC\_ETH2\_COL** 0x42A100B4U, 0x4U, 0x42-A1079CU, 0x1U, 0x42A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_GPIO3\_IO09** 0x42A100B4U, 0x5U, 0, 0, 0x42A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_QTIMER4\_TIMER1** 0x42A100B4U, 0x6U, 0x42-A10880U, 0x0U, 0x42A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B1\_41\_NETC\_PINMUX\_ETH0\_RX\_ER** 0x42A100B4U, 0x9U, 0x42A107E0U, 0x0U, 0x42A102FCU
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_SEMC\_DATA16** 0x42A100B8U, 0x0U, 0, 0, 0x42-A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_CCM\_ENET\_REF\_CLK\_25M** 0x42A100B8U, 0x1U, 0, 0, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_QTIMER5\_TIMER1** 0x42A100B8U, 0x2U, 0x42-A1088CU, 0x0U, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_NETC\_EMDC** 0x42A100B8U, 0x3U, 0, 0, 0x42-A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_NETC\_PINMUX\_ETH0\_RX\_CLK** 0x42A100B8U, 0x4U, 0x42A107D8U, 0x0U, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_GPIO3\_IO10** 0x42A100B8U, 0x5U, 0, 0, 0x42A10300-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_XBAR1\_XBAR\_INOUT20** 0x42A100B8U, 0x6U, 0x42A10948U, 0x0U, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_LPSPi5\_SCK** 0x42A100B8U, 0x8U, 0x42A10644U, 0x1U, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_LPI2C3\_SCL** 0x42A100B8U, 0x9U, 0x42A105E8U, 0x0U, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_FLEXPWM3\_PWMA00** 0x42A100B8U, 0xAU, 0x42-A10524U, 0x1U, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_00\_ECAT\_RX\_CLK\_0** 0x42A100B8U, 0xCU, 0x42A104-ACU, 0x1U, 0x42A10300U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_SEMC\_DATA17** 0x42A100BCU, 0x0U, 0, 0, 0x42-A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_USDHC2\_CD\_B** 0x42A100BCU, 0x1U, 0x42A1092C-U, 0x0U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_QTIMER6\_TIMER1** 0x42A100BCU, 0x2U, 0x42-A10898U, 0x0U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_NETC\_EMDIO** 0x42A100BCU, 0x3U, 0x42A10798U,

- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_NETC\_PINMUX\_ETH0\_RXD02** 0x42A100BCU,  
0x4U, 0x42A107ECU, 0x0U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_GPIO3\_IO11** 0x42A100BCU, 0x5U, 0, 0, 0x42-  
A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_XBAR1\_XBAR\_INOUT21** 0x42A100BCU, 0x6U,  
0x42A1094CU, 0x0U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_LPSPIS\_PCS0** 0x42A100BCU, 0x8U, 0x42A10634U,  
0x1U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_LPI2C3\_SDA** 0x42A100BCU, 0x9U, 0x42A105ECU,  
0x0U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_FLEXPWM3\_PWMB00** 0x42A100BCU, 0xAU, 0x42-  
A10534U, 0x1U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_01\_ECAT\_RX\_DATA2\_0** 0x42A100BCU, 0xCU, 0x42-  
A104C4U, 0x1U, 0x42A10304U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_SEMC\_DATA18** 0x42A100C0U, 0x0U, 0, 0, 0x42-  
A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_USDHC2\_WP** 0x42A100C0U, 0x1U, 0x42A10930U,  
0x0U, 0x42A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_QTIMER7\_TIMER1** 0x42A100C0U, 0x2U, 0x42-  
A108A4U, 0x0U, 0x42A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_NETC\_PINMUX\_ETH0\_RXD03** 0x42A100C0U, 0x4-  
U, 0x42A107F0U, 0x0U, 0x42A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_GPIO3\_IO12** 0x42A100C0U, 0x5U, 0, 0, 0x42A10308-  
U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_XBAR1\_XBAR\_INOUT22** 0x42A100C0U, 0x6U,  
0x42A10950U, 0x0U, 0x42A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_LPSPIS\_SDO** 0x42A100C0U, 0x8U, 0x42A1064CU,  
0x1U, 0x42A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_CCM\_CLKO1** 0x42A100C0U, 0x9U, 0, 0, 0x42-  
A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_FLEXPWM3\_PWMA01** 0x42A100C0U, 0xAU, 0x42-  
A10528U, 0x1U, 0x42A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_02\_ECAT\_RX\_DATA3\_0** 0x42A100C0U, 0xCU, 0x42-  
A104CCU, 0x1U, 0x42A10308U
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_SEMC\_DATA19** 0x42A100C4U, 0x0U, 0, 0, 0x42-  
A1030CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_USDHC2\_VSELECT** 0x42A100C4U, 0x1U, 0, 0,  
0x42A1030CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_QTIMER8\_TIMER1** 0x42A100C4U, 0x2U, 0x42-  
A108ACU, 0x0U, 0x42A1030CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_NETC\_PINMUX\_ETH0\_TXD02** 0x42A100C4U, 0x4-  
U, 0, 0, 0x42A1030CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_GPIO3\_IO13** 0x42A100C4U, 0x5U, 0, 0, 0x42A1030-  
CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_XBAR1\_XBAR\_INOUT23** 0x42A100C4U, 0x6U,  
0x42A10954U, 0x0U, 0x42A1030CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_LPSPIS\_SDI** 0x42A100C4U, 0x8U, 0x42A10648U,  
0x1U, 0x42A1030CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_NETC\_ETH3\_CRS** 0x42A100C4U, 0x9U, 0x42A107-

- B0U, 0x1U, 0x42A1030CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_03\_FLEXPWM3\_PWMB01** 0x42A100C4U, 0xAU, 0x42-A10538U, 0x1U, 0x42A1030CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_03\_ECAT\_TX\_DATA2\_0** 0x42A100C4U, 0x0U, 0, 0, 0x42-A1030CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_SEMC\_DATA20** 0x42A100C8U, 0x0U, 0, 0, 0x42-A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_USDHC2\_RESET\_B** 0x42A100C8U, 0x1U, 0, 0, 0x42-A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_SAI2\_MCLK** 0x42A100C8U, 0x2U, 0, 0, 0x42A10310-U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_NETC\_PINMUX\_ETH0\_TXD03** 0x42A100C8U, 0x4-U, 0, 0, 0x42A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_GPIO3\_IO14** 0x42A100C8U, 0x5U, 0, 0, 0x42A10310-U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_XBAR1\_XBAR\_INOUT24** 0x42A100C8U, 0x6U, 0x42A10958U, 0x0U, 0x42A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_LPSPi3\_SCK** 0x42A100C8U, 0x8U, 0x42A10618U, 0x0U, 0x42A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_NETC\_ETH3\_COL** 0x42A100C8U, 0x9U, 0x42A107-ACU, 0x1U, 0x42A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_FLEXPWM3\_PWMB02** 0x42A100C8U, 0xAU, 0x42-A1053CU, 0x1U, 0x42A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_04\_ECAT\_TX\_DATA3\_0** 0x42A100C8U, 0x0U, 0, 0, 0x42A10310U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_SEMC\_DATA21** 0x42A100CCU, 0x0U, 0, 0, 0x42-A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_NETC\_ETH4\_SLV\_MDC** 0x42A100CCU, 0x1U, 0x42A107C4U, 0x1U, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_SAI2\_RX\_SYNC** 0x42A100CCU, 0x2U, 0, 0, 0x42-A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_NETC\_PINMUX\_ETH0\_TXD00** 0x42A100CCU, 0x3U, 0, 0, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_NETC\_ETH4\_CRS** 0x42A100CCU, 0x4U, 0x42A107-C0U, 0x1U, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_GPIO3\_IO15** 0x42A100CCU, 0x5U, 0, 0, 0x42-A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_XBAR1\_XBAR\_INOUT25** 0x42A100CCU, 0x6U, 0x42A1095CU, 0x0U, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_LPSPi3\_PCS0** 0x42A100CCU, 0x8U, 0x42A10608U, 0x0U, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_NETC\_PINMUX\_ETH3\_TXD00** 0x42A100CCU, 0x9U, 0, 0, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_FLEXPWM3\_PWMA02** 0x42A100CCU, 0xAU, 0x42-A1052CU, 0x1U, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_05\_ECAT\_TX\_DATA0\_0** 0x42A100CCU, 0x0U, 0, 0, 0x42A10314U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_SEMC\_DATA22** 0x42A100D0U, 0x0U, 0, 0, 0x42-A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_FLEXPWM3\_PWMB03** 0x42A100D0U, 0xAU, 0x42-

- A10540U, 0x1U, 0x42A10318U
- #define **IOMUXC\_GPIO\_EMC\_B2\_06\_NETC\_ETH4\_SLV\_MDIO** 0x42A100D0U, 0x1U, 0x42A107C8U, 0x1U, 0x42A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_SAI2\_RX\_BCLK** 0x42A100D0U, 0x2U, 0, 0, 0x42-A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_ECAT\_TX\_DATA1\_0** 0x42A100D0U, 0xCU, 0, 0, 0x42A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_NETC\_PINMUX\_ETH0\_TXD01** 0x42A100D0U, 0x3-U, 0, 0, 0x42A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_NETC\_ETH4\_COL** 0x42A100D0U, 0x4U, 0x42A107-BCU, 0x1U, 0x42A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_GPIO3\_IO16** 0x42A100D0U, 0x5U, 0, 0, 0x42A10318-U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_XBAR1\_XBAR\_INOUT26** 0x42A100D0U, 0x6U, 0x42A10960U, 0x0U, 0x42A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_LPSPI3\_SDO** 0x42A100D0U, 0x8U, 0x42A10620U, 0x0U, 0x42A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_06\_NETC\_PINMUX\_ETH3\_TXD01** 0x42A100D0U, 0x9-U, 0, 0, 0x42A10318U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_SEMC\_DATA23** 0x42A100D4U, 0x0U, 0, 0, 0x42-A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_NETC\_PINMUX\_ETH4\_TX\_ER** 0x42A100D4U, 0x1U, 0, 0, 0x42A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_SAI2\_RX\_DATA** 0x42A100D4U, 0x2U, 0, 0, 0x42-A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_NETC\_PINMUX\_ETH0\_TX\_EN** 0x42A100D4U, 0x3U, 0, 0, 0x42A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_GPIO3\_IO17** 0x42A100D4U, 0x5U, 0, 0, 0x42A1031-CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_XBAR1\_XBAR\_INOUT27** 0x42A100D4U, 0x6U, 0x42A10964U, 0x0U, 0x42A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_LPSPI3\_SDI** 0x42A100D4U, 0x8U, 0x42A1061CU, 0x0U, 0x42A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_NETC\_PINMUX\_ETH3\_TX\_EN** 0x42A100D4U, 0x9U, 0, 0, 0x42A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_FLEXPWM3\_PWMA03** 0x42A100D4U, 0xAU, 0x42-A10530U, 0x1U, 0x42A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_07\_ECAT\_TX\_EN\_0** 0x42A100D4U, 0xCU, 0, 0, 0x42-A1031CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_08\_SEMC\_DM02** 0x42A100D8U, 0x0U, 0, 0, 0x42-A10320U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_08\_NETC\_PINMUX\_ETH4\_RX\_CLK** 0x42A100D8U, 0x1U, 0x42A10838U, 0x1U, 0x42A10320U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_08\_SAI2\_TX\_DATA** 0x42A100D8U, 0x2U, 0, 0, 0x42-A10320U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_08\_NETC\_PINMUX\_ETH0\_TX\_CLK** 0x42A100D8U, 0x3U, 0x42A107F4U, 0x1U, 0x42A10320U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_08\_GPIO3\_IO18** 0x42A100D8U, 0x5U, 0, 0, 0x42A10320-U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_08\_XBAR1\_XBAR\_INOUT28** 0x42A100D8U, 0x6U,

- 0x42A10968U, 0x0U, 0x42A10320U
- #define **IOMUXC\_GPIO\_EMC\_B2\_08\_LPSPi3\_PCS3** 0x42A100D8U, 0x8U, 0x42A10614U, 0x0U, 0x42A10320U
- #define **IOMUXC\_GPIO\_EMC\_B2\_08\_NETC\_PINMUX\_ETH3\_TX\_CLK** 0x42A100D8U, 0x9U, 0x42A10834U, 0x1U, 0x42A10320U
- #define **IOMUXC\_GPIO\_EMC\_B2\_08\_CCM\_CLKO2** 0x42A100D8U, 0xAU, 0, 0, 0x42-A10320U
- #define **IOMUXC\_GPIO\_EMC\_B2\_08\_ECAT\_TX\_CLK\_0** 0x42A100D8U, 0xCU, 0x42A104-E4U, 0x1U, 0x42A10320U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_QTIMER1\_TIMER0** 0x42A100DCU, 0xAU, 0x42-A10858U, 0x1U, 0x42A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_ECAT\_RX\_DATA0\_0** 0x42A100DCU, 0xCU, 0x42-A104B4U, 0x1U, 0x42A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_SEMC\_DATA24** 0x42A100DCU, 0x0U, 0, 0, 0x42-A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_NETC\_PINMUX\_ETH4\_RXD03** 0x42A100DCU, 0x1U, 0x42A10850U, 0x1U, 0x42A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_SAI2\_TX\_BCLK** 0x42A100DCU, 0x2U, 0, 0, 0x42-A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_NETC\_PINMUX\_ETH0\_RXD00** 0x42A100DCU, 0x3U, 0x42A107E4U, 0x1U, 0x42A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_GPIO3\_IO19** 0x42A100DCU, 0x5U, 0, 0, 0x42-A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_XBAR1\_XBAR\_INOUT29** 0x42A100DCU, 0x6U, 0x42A1096CU, 0x0U, 0x42A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_LPSPi3\_PCS2** 0x42A100DCU, 0x8U, 0x42A10610U, 0x0U, 0x42A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_09\_NETC\_PINMUX\_ETH3\_RXD00** 0x42A100DCU, 0x9U, 0x42A10824U, 0x1U, 0x42A10324U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_QTIMER1\_TIMER1** 0x42A100E0U, 0xAU, 0x42-A1085CU, 0x1U, 0x42A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_ECAT\_RX\_DATA1\_0** 0x42A100E0U, 0xCU, 0x42-A104BCU, 0x1U, 0x42A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_SEMC\_DATA25** 0x42A100E0U, 0x0U, 0, 0, 0x42-A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_NETC\_PINMUX\_ETH4\_RXD02** 0x42A100E0U, 0x1-U, 0x42A1084CU, 0x1U, 0x42A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_SAI2\_TX\_SYNC** 0x42A100E0U, 0x2U, 0, 0, 0x42-A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_NETC\_PINMUX\_ETH0\_RXD01** 0x42A100E0U, 0x3-U, 0x42A107E8U, 0x1U, 0x42A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_GPIO3\_IO20** 0x42A100E0U, 0x5U, 0, 0, 0x42A10328-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_XBAR1\_XBAR\_INOUT30** 0x42A100E0U, 0x6U, 0x42A10970U, 0x0U, 0x42A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_LPSPi3\_PCS1** 0x42A100E0U, 0x8U, 0x42A1060CU, 0x0U, 0x42A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_10\_NETC\_PINMUX\_ETH3\_RXD01** 0x42A100E0U, 0x9-U, 0x42A10828U, 0x1U, 0x42A10328U
- #define **IOMUXC\_GPIO\_EMC\_B2\_11\_SEMC\_DATA26** 0x42A100E4U, 0x0U, 0, 0, 0x42-

- A1032CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_11\_NETC\_PINMUX\_ETH4\_TXD03** 0x42A100E4U, 0x1-U, 0, 0, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_SPDIF\_OUT** 0x42A100E4U, 0x2U, 0, 0, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_NETC\_PINMUX\_ETH0\_RX\_DV** 0x42A100E4U, 0x3U, 0x42A107DCU, 0x1U, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_LPSPIS5\_PCS3** 0x42A100E4U, 0x4U, 0x42A10640U, 0x0U, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_GPIO3\_IO21** 0x42A100E4U, 0x5U, 0, 0, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_XBAR1\_XBAR\_INOUT31** 0x42A100E4U, 0x6U, 0x42A10974U, 0x0U, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_SAI3\_RX\_SYNC** 0x42A100E4U, 0x8U, 0, 0, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_NETC\_PINMUX\_ETH3\_RX\_DV** 0x42A100E4U, 0x9U, 0x42A1081CU, 0x1U, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_QTIMER1\_TIMER2** 0x42A100E4U, 0xAU, 0x42A10860U, 0x0U, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_11\_ECAT\_RX\_DV\_0** 0x42A100E4U, 0xCU, 0x42A104D4U, 0x1U, 0x42A1032CU
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_SEMC\_DATA27** 0x42A100E8U, 0x0U, 0, 0, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_NETC\_PINMUX\_ETH4\_TXD02** 0x42A100E8U, 0x1-U, 0, 0, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_SPDIF\_IN** 0x42A100E8U, 0x2U, 0x42A10908U, 0x0U, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_NETC\_PINMUX\_ETH0\_RX\_ER** 0x42A100E8U, 0x3U, 0x42A107E0U, 0x1U, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_LPSPIS5\_PCS2** 0x42A100E8U, 0x4U, 0x42A1063CU, 0x0U, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_GPIO3\_IO22** 0x42A100E8U, 0x5U, 0, 0, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_XBAR1\_XBAR\_INOUT32** 0x42A100E8U, 0x6U, 0x42A10978U, 0x0U, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_SAI3\_RX\_BCLK** 0x42A100E8U, 0x8U, 0, 0, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_NETC\_PINMUX\_ETH3\_RX\_ER** 0x42A100E8U, 0x9U, 0x42A10820U, 0x1U, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_QTIMER1\_TIMER3** 0x42A100E8U, 0xAU, 0, 0, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_12\_ECAT\_PT0\_RX\_ER** 0x42A100E8U, 0xCU, 0x42A104DCU, 0x1U, 0x42A10330U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_13\_QTIMER2\_TIMER0** 0x42A100ECU, 0xAU, 0x42A10864U, 0x1U, 0x42A10334U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_13\_ECAT\_TX\_DATA0\_1** 0x42A100ECU, 0xCU, 0, 0, 0x42A10334U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_13\_SEMC\_DATA28** 0x42A100ECU, 0x0U, 0, 0, 0x42A10334U
  - #define **IOMUXC\_GPIO\_EMC\_B2\_13\_NETC\_PINMUX\_ETH4\_TXD00** 0x42A100ECU,

- 0x1U, 0, 0, 0x42A10334U
- #define **IOMUXC\_GPIO\_EMC\_B2\_13\_LPUART11\_TX** 0x42A100ECU, 0x2U, 0x42A10678U, 0x0U, 0x42A10334U
- #define **IOMUXC\_GPIO\_EMC\_B2\_13\_NETC\_PINMUX\_ETH0\_RXD03** 0x42A100ECU, 0x3U, 0, 0, 0x42A10334U
- #define **IOMUXC\_GPIO\_EMC\_B2\_13\_LPSPS15\_PCS1** 0x42A100ECU, 0x4U, 0x42A10638U, 0x1U, 0x42A10334U
- #define **IOMUXC\_GPIO\_EMC\_B2\_13\_GPIO3\_IO23** 0x42A100ECU, 0x5U, 0, 0, 0x42A10334-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_13\_XBAR1\_XBAR\_INOUT33** 0x42A100ECU, 0x6U, 0x42A1097CU, 0x0U, 0x42A10334U
- #define **IOMUXC\_GPIO\_EMC\_B2\_13\_SAI3\_RX\_DATA** 0x42A100ECU, 0x8U, 0, 0, 0x42A10334U
- #define **IOMUXC\_GPIO\_EMC\_B2\_13\_NETC\_PINMUX\_ETH3\_RXD03** 0x42A100ECU, 0x9U, 0, 0, 0x42A10334U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_SEMC\_DATA29** 0x42A100F0U, 0x0U, 0, 0, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_NETC\_PINMUX\_ETH4\_RXD01** 0x42A100F0U, 0x1U, 0, 0, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_LPUART11\_RX** 0x42A100F0U, 0x2U, 0x42A10674U, 0x0U, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_NETC\_PINMUX\_ETH0\_RXD02** 0x42A100F0U, 0x3U, 0, 0, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_LPUART5\_DSR\_B** 0x42A100F0U, 0x4U, 0x42A10694U, 0x0U, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_GPIO3\_IO24** 0x42A100F0U, 0x5U, 0, 0, 0x42A10338-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_XBAR1\_XBAR\_INOUT34** 0x42A100F0U, 0x6U, 0x42A10980U, 0x0U, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_SAI3\_TX\_DATA** 0x42A100F0U, 0x8U, 0, 0, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_NETC\_PINMUX\_ETH3\_RXD02** 0x42A100F0U, 0x9U, 0, 0, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_QTIMER2\_TIMER1** 0x42A100F0U, 0xAU, 0x42A10868U, 0x1U, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_14\_ECAT\_TX\_DATA1\_1** 0x42A100F0U, 0xCU, 0, 0, 0x42A10338U
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_SEMC\_DATA30** 0x42A100F4U, 0x0U, 0, 0, 0x42A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_NETC\_PINMUX\_ETH4\_RX\_EN** 0x42A100F4U, 0x1U, 0, 0, 0x42A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_LPUART11\_CTS\_B** 0x42A100F4U, 0x2U, 0, 0, 0x42A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_NETC\_PINMUX\_ETH0\_RX\_CLK** 0x42A100F4U, 0x3U, 0x42A107D8U, 0x1U, 0x42A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_LPUART5\_DCD\_B** 0x42A100F4U, 0x4U, 0x42A10690U, 0x0U, 0x42A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_GPIO3\_IO25** 0x42A100F4U, 0x5U, 0, 0, 0x42A1033-CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_XBAR1\_XBAR\_INOUT35** 0x42A100F4U, 0x6U,

- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_SAI3\_TX\_BCLK** 0x42A100F4U, 0x8U, 0, 0, 0x42-A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_NETC\_PINMUX\_ETH3\_RX\_CLK** 0x42A100F4U, 0x9U, 0x42A10818U, 0x1U, 0x42A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_QTIMER2\_TIMER2** 0x42A100F4U, 0xAU, 0x42-A1086CU, 0x0U, 0x42A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_15\_ECAT\_TX\_EN\_1** 0x42A100F4U, 0xCU, 0, 0, 0x42-A1033CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_QTIMER2\_TIMER3** 0x42A100F8U, 0xAU, 0, 0, 0x42A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_ECAT\_TX\_CLK\_1** 0x42A100F8U, 0xCU, 0x42A104-E8U, 0x1U, 0x42A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_SEMC\_DATA31** 0x42A100F8U, 0x0U, 0, 0, 0x42-A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_NETC\_PINMUX\_ETH4\_TX\_CLK** 0x42A100F8U, 0x1U, 0x42A10854U, 0x1U, 0x42A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_LPUART11\_RTS\_B** 0x42A100F8U, 0x2U, 0, 0, 0x42-A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_NETC\_PINMUX\_ETH0\_RXD02** 0x42A100F8U, 0x3-U, 0x42A107ECU, 0x1U, 0x42A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_LPUART5\_DTR\_B** 0x42A100F8U, 0x4U, 0, 0, 0x42-A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_GPIO3\_IO26** 0x42A100F8U, 0x5U, 0, 0, 0x42A10340-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_XBAR1\_XBAR\_INOUT14** 0x42A100F8U, 0x6U, 0x42A10934U, 0x1U, 0x42A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_SAI3\_TX\_SYNC** 0x42A100F8U, 0x8U, 0, 0, 0x42-A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_16\_NETC\_PINMUX\_ETH3\_RXD02** 0x42A100F8U, 0x9-U, 0x42A1082CU, 0x1U, 0x42A10340U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_SEMC\_DM03** 0x42A100FCU, 0x0U, 0, 0, 0x42-A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_NETC\_PINMUX\_ETH4\_RXD00** 0x42A100FCU, 0x1U, 0x42A10844U, 0x1U, 0x42A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_LPUART5\_TX** 0x42A100FCU, 0x2U, 0x42A106A0U, 0x2U, 0x42A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_NETC\_PINMUX\_ETH0\_RXD03** 0x42A100FCU, 0x3U, 0x42A107F0U, 0x1U, 0x42A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_GPIO3\_IO27** 0x42A100FCU, 0x5U, 0, 0, 0x42A10344-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_XBAR1\_XBAR\_INOUT15** 0x42A100FCU, 0x6U, 0x42A10938U, 0x1U, 0x42A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_SAI3\_MCLK** 0x42A100FCU, 0x8U, 0, 0, 0x42A10344-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_NETC\_PINMUX\_ETH3\_RXD03** 0x42A100FCU, 0x9U, 0x42A10830U, 0x1U, 0x42A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_QTIMER3\_TIMER0** 0x42A100FCU, 0xAU, 0x42-A10870U, 0x1U, 0x42A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_17\_ECAT\_RX\_DATA0\_1** 0x42A100FCU, 0xCU, 0x42-

- A104B8U, 0x1U, 0x42A10344U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_SEMC\_DQS4** 0x42A10100U, 0x0U, 0, 0, 0x42A10348-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_NETC\_PINMUX\_ETH4\_RXD01** 0x42A10100U, 0x1U, 0x42A10848U, 0x1U, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_LPUART5\_RX** 0x42A10100U, 0x2U, 0x42A1069CU, 0x2U, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_NETC\_PINMUX\_ETH0\_TX\_ER** 0x42A10100U, 0x3U, 0, 0, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_GPIO3\_IO28** 0x42A10100U, 0x5U, 0, 0, 0x42A10348-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_XBAR1\_XBAR\_INOUT16** 0x42A10100U, 0x6U, 0, 0, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_EWM\_EWM\_OUT\_B** 0x42A10100U, 0x8U, 0, 0, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_NETC\_PINMUX\_ETH3\_TX\_ER** 0x42A10100U, 0x9U, 0, 0, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_QTIMER3\_TIMER1** 0x42A10100U, 0xAU, 0x42-A10874U, 0x1U, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_18\_ECAT\_RX\_DATA1\_1** 0x42A10100U, 0x1CU, 0x42-A104C0U, 0x1U, 0x42A10348U
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_QTIMER3\_TIMER2** 0x42A10104U, 0xAU, 0x42-A10878U, 0x0U, 0x42A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_ECAT\_RX\_DV\_1** 0x42A10104U, 0x1CU, 0x42A104-D8U, 0x1U, 0x42A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_SEMC\_CLKX00** 0x42A10104U, 0x0U, 0, 0, 0x42-A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_NETC\_PINMUX\_ETH4\_RX\_DV** 0x42A10104U, 0x1U, 0x42A1083CU, 0x1U, 0x42A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_LPUART5\_CTS\_B** 0x42A10104U, 0x2U, 0x42A1068-CU, 0x1U, 0x42A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_NETC\_ETH0 CRS** 0x42A10104U, 0x3U, 0, 0, 0x42-A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_NETC\_EMDC** 0x42A10104U, 0x4U, 0, 0, 0x42A1034-CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_GPIO3\_IO29** 0x42A10104U, 0x5U, 0, 0, 0x42A1034-CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_XBAR1\_XBAR\_INOUT36** 0x42A10104U, 0x6U, 0x42A10988U, 0x0U, 0x42A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_LPI2C3\_SCL** 0x42A10104U, 0x8U, 0x42A105E8U, 0x1U, 0x42A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_19\_NETC\_ETH3\_SLV\_MDC** 0x42A10104U, 0x9U, 0x42A107B4U, 0x2U, 0x42A1034CU
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_QTIMER3\_TIMER3** 0x42A10108U, 0xAU, 0, 0, 0x42A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_ECAT\_RX\_ER\_1** 0x42A10108U, 0x1CU, 0x42A104E0-U, 0x1U, 0x42A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_SEMC\_CLKX01** 0x42A10108U, 0x0U, 0, 0, 0x42-A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_NETC\_PINMUX\_ETH4\_RX\_ER** 0x42A10108U, 0x1-

- U, 0x42A10840U, 0x1U, 0x42A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_LPUART5\_RTS\_B** 0x42A10108U, 0x2U, 0, 0, 0x42-A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_NETC\_ETH0\_COL** 0x42A10108U, 0x3U, 0, 0, 0x42-A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_NETC\_EMDIO** 0x42A10108U, 0x4U, 0x42A10798U, 0x3U, 0x42A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_GPIO3\_IO30** 0x42A10108U, 0x5U, 0, 0, 0x42A10350-U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_XBAR1\_XBAR\_INOUT37** 0x42A10108U, 0x6U, 0x42A1098CU, 0x0U, 0x42A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_LPI2C3\_SDA** 0x42A10108U, 0x8U, 0x42A105ECU, 0x1U, 0x42A10350U
- #define **IOMUXC\_GPIO\_EMC\_B2\_20\_NETC\_ETH3\_SLV\_MDIO** 0x42A10108U, 0x9U, 0x42A107B8U, 0x2U, 0x42A10350U
- #define **IOMUXC\_GPIO\_AD\_00\_CAN2\_TX** 0x42A1010CU, 0x1U, 0, 0, 0x42A10354U
- #define **IOMUXC\_GPIO\_AD\_00\_MIC\_CLK** 0x42A1010CU, 0x2U, 0, 0, 0x42A10354U
- #define **IOMUXC\_GPIO\_AD\_00\_GPT2\_CAPTURE1** 0x42A1010CU, 0x3U, 0, 0, 0x42A10354-U
- #define **IOMUXC\_GPIO\_AD\_00\_FLEXPWM1\_PWMA00** 0x42A1010CU, 0x4U, 0x42A104-F4U, 0x2U, 0x42A10354U
- #define **IOMUXC\_GPIO\_AD\_00\_GPIO4\_IO00** 0x42A1010CU, 0x5U, 0, 0, 0x42A10354U
- #define **IOMUXC\_GPIO\_AD\_00\_SINC1\_MOD\_CLK0** 0x42A1010CU, 0x6U, 0, 0, 0x42-A10354U
- #define **IOMUXC\_GPIO\_AD\_00\_FLEXIO2\_FLEXIO00** 0x42A1010CU, 0x8U, 0, 0, 0x42-A10354U
- #define **IOMUXC\_GPIO\_AD\_00\_QTIMER4\_TIMER0** 0x42A1010CU, 0x9U, 0x42A1087CU, 0x1U, 0x42A10354U
- #define **IOMUXC\_GPIO\_AD\_01\_CAN2\_RX** 0x42A10110U, 0x1U, 0x42A104A4U, 0x0U, 0x42A10358U
- #define **IOMUXC\_GPIO\_AD\_01\_MIC\_BITSTREAM00** 0x42A10110U, 0x2U, 0x42A106D0U, 0x0U, 0x42A10358U
- #define **IOMUXC\_GPIO\_AD\_01\_GPT2\_CAPTURE2** 0x42A10110U, 0x3U, 0, 0, 0x42A10358-U
- #define **IOMUXC\_GPIO\_AD\_01\_FLEXPWM1\_PWMB00** 0x42A10110U, 0x4U, 0x42A10500-U, 0x2U, 0x42A10358U
- #define **IOMUXC\_GPIO\_AD\_01\_GPIO4\_IO01** 0x42A10110U, 0x5U, 0, 0, 0x42A10358U
- #define **IOMUXC\_GPIO\_AD\_01\_SINC1\_MOD\_CLK1** 0x42A10110U, 0x6U, 0, 0, 0x42-A10358U
- #define **IOMUXC\_GPIO\_AD\_01\_FLEXIO2\_FLEXIO01** 0x42A10110U, 0x8U, 0, 0, 0x42-A10358U
- #define **IOMUXC\_GPIO\_AD\_01\_QTIMER4\_TIMER1** 0x42A10110U, 0x9U, 0x42A10880U, 0x1U, 0x42A10358U
- #define **IOMUXC\_GPIO\_AD\_02\_MIC\_BITSTREAM01** 0x42A10114U, 0x2U, 0x42A106D4U, 0x0U, 0x42A1035CU
- #define **IOMUXC\_GPIO\_AD\_02\_GPT2\_COMPARE1** 0x42A10114U, 0x3U, 0, 0, 0x42A1035-CU
- #define **IOMUXC\_GPIO\_AD\_02\_FLEXPWM1\_PWMA01** 0x42A10114U, 0x4U, 0x42A104-F8U, 0x1U, 0x42A1035CU
- #define **IOMUXC\_GPIO\_AD\_02\_GPIO4\_IO02** 0x42A10114U, 0x5U, 0, 0, 0x42A1035CU

- #define **IOMUXC\_GPIO\_AD\_02\_SINC1\_MOD\_CLK2** 0x42A10114U, 0x6U, 0, 0, 0x42A1035-CU
- #define **IOMUXC\_GPIO\_AD\_02\_FLEXIO2\_FLEXIO02** 0x42A10114U, 0x8U, 0, 0, 0x42-A1035CU
- #define **IOMUXC\_GPIO\_AD\_02\_QTIMER4\_TIMER2** 0x42A10114U, 0x9U, 0x42A10884U, 0x0U, 0x42A1035CU
- #define **IOMUXC\_GPIO\_AD\_03\_MIC\_BITSTREAM02** 0x42A10118U, 0x2U, 0x42A106D8U, 0x0U, 0x42A10360U
- #define **IOMUXC\_GPIO\_AD\_03\_GPT2\_COMPARE2** 0x42A10118U, 0x3U, 0, 0, 0x42-A10360U
- #define **IOMUXC\_GPIO\_AD\_03\_FLEXPWM1\_PWMB01** 0x42A10118U, 0x4U, 0x42A10504-U, 0x1U, 0x42A10360U
- #define **IOMUXC\_GPIO\_AD\_03\_GPIO4\_IO03** 0x42A10118U, 0x5U, 0, 0, 0x42A10360U
- #define **IOMUXC\_GPIO\_AD\_03\_SINC1\_EMCLK00** 0x42A10118U, 0x6U, 0x42A108E4U, 0x0U, 0x42A10360U
- #define **IOMUXC\_GPIO\_AD\_03\_FLEXIO2\_FLEXIO03** 0x42A10118U, 0x8U, 0, 0, 0x42-A10360U
- #define **IOMUXC\_GPIO\_AD\_03\_QTIMER4\_TIMER3** 0x42A10118U, 0x9U, 0, 0, 0x42-A10360U
- #define **IOMUXC\_GPIO\_AD\_04\_MIC\_BITSTREAM03** 0x42A1011CU, 0x2U, 0x42A106DC-U, 0x0U, 0x42A10364U
- #define **IOMUXC\_GPIO\_AD\_04\_GPT2\_COMPARE3** 0x42A1011CU, 0x3U, 0, 0, 0x42-A10364U
- #define **IOMUXC\_GPIO\_AD\_04\_FLEXPWM1\_PWMB02** 0x42A1011CU, 0x4U, 0x42-A10508U, 0x1U, 0x42A10364U
- #define **IOMUXC\_GPIO\_AD\_04\_GPIO4\_IO04** 0x42A1011CU, 0x5U, 0, 0, 0x42A10364U
- #define **IOMUXC\_GPIO\_AD\_04\_SINC1\_EMBIT00** 0x42A1011CU, 0x6U, 0x42A108D4U, 0x0U, 0x42A10364U
- #define **IOMUXC\_GPIO\_AD\_04\_FLEXIO2\_FLEXIO04** 0x42A1011CU, 0x8U, 0, 0, 0x42-A10364U
- #define **IOMUXC\_GPIO\_AD\_04\_QTIMER5\_TIMER0** 0x42A1011CU, 0x9U, 0x42A10888U, 0x1U, 0x42A10364U
- #define **IOMUXC\_GPIO\_AD\_05\_GPT2\_CLK** 0x42A10120U, 0x3U, 0, 0, 0x42A10368U
- #define **IOMUXC\_GPIO\_AD\_05\_FLEXPWM1\_PWMA02** 0x42A10120U, 0x4U, 0x42A104F-CU, 0x1U, 0x42A10368U
- #define **IOMUXC\_GPIO\_AD\_05\_GPIO4\_IO05** 0x42A10120U, 0x5U, 0, 0, 0x42A10368U
- #define **IOMUXC\_GPIO\_AD\_05\_SINC1\_EMCLK01** 0x42A10120U, 0x6U, 0x42A108E8U, 0x0U, 0x42A10368U
- #define **IOMUXC\_GPIO\_AD\_05\_CCM\_ENET\_REF\_CLK\_25M** 0x42A10120U, 0x7U, 0, 0, 0x42A10368U
- #define **IOMUXC\_GPIO\_AD\_05\_FLEXIO2\_FLEXIO05** 0x42A10120U, 0x8U, 0, 0, 0x42-A10368U
- #define **IOMUXC\_GPIO\_AD\_05\_QTIMER5\_TIMER1** 0x42A10120U, 0x9U, 0x42A1088CU, 0x1U, 0x42A10368U
- #define **IOMUXC\_GPIO\_AD\_06\_USB\_OTG2\_OC** 0x42A10124U, 0x0U, 0x42A10914U, 0x0U, 0x42A1036CU
- #define **IOMUXC\_GPIO\_AD\_06\_CAN3\_TX** 0x42A10124U, 0x1U, 0, 0, 0x42A1036CU
- #define **IOMUXC\_GPIO\_AD\_06\_FLEXPWM1\_PWMX00** 0x42A10124U, 0x4U, 0, 0, 0x42-A1036CU
- #define **IOMUXC\_GPIO\_AD\_06\_GPIO4\_IO06** 0x42A10124U, 0x5U, 0, 0, 0x42A1036CU

- #define **IOMUXC\_GPIO\_AD\_06\_SINC1\_EMBIT01** 0x42A10124U, 0x6U, 0x42A108D8U, 0x0U, 0x42A1036CU
- #define **IOMUXC\_GPIO\_AD\_06\_FLEXIO2\_FLEXIO06** 0x42A10124U, 0x8U, 0, 0, 0x42-A1036CU
- #define **IOMUXC\_GPIO\_AD\_06\_QTIMER5\_TIMER2** 0x42A10124U, 0x9U, 0x42A10890U, 0x0U, 0x42A1036CU
- #define **IOMUXC\_GPIO\_AD\_07\_USB\_OTG2\_PWR** 0x42A10128U, 0x0U, 0, 0, 0x42A10370U
- #define **IOMUXC\_GPIO\_AD\_07\_CAN3\_RX** 0x42A10128U, 0x1U, 0x42A104A8U, 0x0U, 0x42A10370U
- #define **IOMUXC\_GPIO\_AD\_07\_FLEXPWM1\_PWMX01** 0x42A10128U, 0x4U, 0, 0, 0x42-A10370U
- #define **IOMUXC\_GPIO\_AD\_07\_GPIO4\_IO07** 0x42A10128U, 0x5U, 0, 0, 0x42A10370U
- #define **IOMUXC\_GPIO\_AD\_07\_SINC1\_EMCLK02** 0x42A10128U, 0x6U, 0x42A108ECU, 0x0U, 0x42A10370U
- #define **IOMUXC\_GPIO\_AD\_07\_FLEXIO2\_FLEXIO07** 0x42A10128U, 0x8U, 0, 0, 0x42-A10370U
- #define **IOMUXC\_GPIO\_AD\_07\_QTIMER5\_TIMER3** 0x42A10128U, 0x9U, 0, 0, 0x42-A10370U
- #define **IOMUXC\_GPIO\_AD\_08\_USBPHY2\_OTG\_ID** 0x42A1012CU, 0x0U, 0x42A10920U, 0x0U, 0x42A10374U
- #define **IOMUXC\_GPIO\_AD\_08\_LPI2C5\_SCL** 0x42A1012CU, 0x1U, 0x42A105F8U, 0x0U, 0x42A10374U
- #define **IOMUXC\_GPIO\_AD\_08\_FLEXPWM1\_PWMX02** 0x42A1012CU, 0x4U, 0, 0, 0x42-A10374U
- #define **IOMUXC\_GPIO\_AD\_08\_GPIO4\_IO08** 0x42A1012CU, 0x5U, 0, 0, 0x42A10374U
- #define **IOMUXC\_GPIO\_AD\_08\_SINC1\_EMBIT02** 0x42A1012CU, 0x6U, 0x42A108DCU, 0x0U, 0x42A10374U
- #define **IOMUXC\_GPIO\_AD\_08\_FLEXIO2\_FLEXIO08** 0x42A1012CU, 0x8U, 0, 0, 0x42-A10374U
- #define **IOMUXC\_GPIO\_AD\_08\_QTIMER6\_TIMER0** 0x42A1012CU, 0x9U, 0x42A10894U, 0x1U, 0x42A10374U
- #define **IOMUXC\_GPIO\_AD\_09\_USBPHY1\_OTG\_ID** 0x42A10130U, 0x0U, 0x42A1091CU, 0x0U, 0x42A10378U
- #define **IOMUXC\_GPIO\_AD\_09\_LPI2C5\_SDA** 0x42A10130U, 0x1U, 0x42A105FCU, 0x0U, 0x42A10378U
- #define **IOMUXC\_GPIO\_AD\_09\_FLEXPWM1\_PWMX03** 0x42A10130U, 0x4U, 0, 0, 0x42-A10378U
- #define **IOMUXC\_GPIO\_AD\_09\_GPIO4\_IO09** 0x42A10130U, 0x5U, 0, 0, 0x42A10378U
- #define **IOMUXC\_GPIO\_AD\_09\_SINC1\_EMCLK03** 0x42A10130U, 0x6U, 0x42A108F0U, 0x0U, 0x42A10378U
- #define **IOMUXC\_GPIO\_AD\_09\_FLEXIO2\_FLEXIO09** 0x42A10130U, 0x8U, 0, 0, 0x42-A10378U
- #define **IOMUXC\_GPIO\_AD\_09\_QTIMER6\_TIMER1** 0x42A10130U, 0x9U, 0x42A10898U, 0x1U, 0x42A10378U
- #define **IOMUXC\_GPIO\_AD\_10\_USB\_OTG1\_PWR** 0x42A10134U, 0x0U, 0, 0, 0x42A1037CU
- #define **IOMUXC\_GPIO\_AD\_10\_FLEXPWM2\_PWMX00** 0x42A10134U, 0x4U, 0, 0, 0x42-A1037CU
- #define **IOMUXC\_GPIO\_AD\_10\_GPIO4\_IO10** 0x42A10134U, 0x5U, 0, 0, 0x42A1037CU
- #define **IOMUXC\_GPIO\_AD\_10\_SINC1\_EMBIT03** 0x42A10134U, 0x6U, 0x42A108E0U, 0x0-U, 0x42A1037CU

- #define **IOMUXC\_GPIO\_AD\_10\_FLEXIO2\_FLEXIO10** 0x42A10134U, 0x8U, 0, 0, 0x42-A1037CU
- #define **IOMUXC\_GPIO\_AD\_10\_QTIMER6\_TIMER2** 0x42A10134U, 0x9U, 0x42A1089CU, 0x0U, 0x42A1037CU
- #define **IOMUXC\_GPIO\_AD\_11\_USB\_OTG1\_OC** 0x42A10138U, 0x0U, 0x42A10918U, 0x0U, 0x42A10380U
- #define **IOMUXC\_GPIO\_AD\_11\_FLEXPWM2\_PWMX01** 0x42A10138U, 0x4U, 0, 0, 0x42-A10380U
- #define **IOMUXC\_GPIO\_AD\_11\_GPIO4\_IO11** 0x42A10138U, 0x5U, 0, 0, 0x42A10380U
- #define **IOMUXC\_GPIO\_AD\_11\_SINC\_FILTER\_GLUE1\_BREAK** 0x42A10138U, 0x6U, 0, 0, 0x42A10380U
- #define **IOMUXC\_GPIO\_AD\_11\_FLEXIO2\_FLEXIO11** 0x42A10138U, 0x8U, 0, 0, 0x42-A10380U
- #define **IOMUXC\_GPIO\_AD\_11\_QTIMER6\_TIMER3** 0x42A10138U, 0x9U, 0, 0, 0x42-A10380U
- #define **IOMUXC\_GPIO\_AD\_12\_SPDIF\_LOCK** 0x42A1013CU, 0x0U, 0, 0, 0x42A10384U
- #define **IOMUXC\_GPIO\_AD\_12\_LPI2C5\_SCLS** 0x42A1013CU, 0x1U, 0, 0, 0x42A10384U
- #define **IOMUXC\_GPIO\_AD\_12\_GPT1\_CAPTURE1** 0x42A1013CU, 0x2U, 0, 0, 0x42A10384-U
- #define **IOMUXC\_GPIO\_AD\_12\_KPP\_ROW07** 0x42A1013CU, 0x3U, 0x42A105E4U, 0x1U, 0x42A10384U
- #define **IOMUXC\_GPIO\_AD\_12\_FLEXPWM2\_PWMX02** 0x42A1013CU, 0x4U, 0, 0, 0x42-A10384U
- #define **IOMUXC\_GPIO\_AD\_12\_GPIO4\_IO12** 0x42A1013CU, 0x5U, 0, 0, 0x42A10384U
- #define **IOMUXC\_GPIO\_AD\_12\_XBAR1\_XBAR\_INOUT18** 0x42A1013CU, 0x6U, 0x42-A10940U, 0x0U, 0x42A10384U
- #define **IOMUXC\_GPIO\_AD\_12\_EWM\_EWM\_OUT\_B** 0x42A1013CU, 0x7U, 0, 0, 0x42-A10384U
- #define **IOMUXC\_GPIO\_AD\_12\_FLEXIO2\_FLEXIO12** 0x42A1013CU, 0x8U, 0, 0, 0x42-A10384U
- #define **IOMUXC\_GPIO\_AD\_13\_SPDIF\_SR\_CLK** 0x42A10140U, 0x0U, 0, 0, 0x42A10388U
- #define **IOMUXC\_GPIO\_AD\_13\_LPI2C5\_SDAS** 0x42A10140U, 0x1U, 0, 0, 0x42A10388U
- #define **IOMUXC\_GPIO\_AD\_13\_GPT1\_CAPTURE2** 0x42A10140U, 0x2U, 0, 0, 0x42A10388-U
- #define **IOMUXC\_GPIO\_AD\_13\_KPP\_COL07** 0x42A10140U, 0x3U, 0x42A105C4U, 0x1U, 0x42A10388U
- #define **IOMUXC\_GPIO\_AD\_13\_FLEXPWM2\_PWMX03** 0x42A10140U, 0x4U, 0, 0, 0x42-A10388U
- #define **IOMUXC\_GPIO\_AD\_13\_GPIO4\_IO13** 0x42A10140U, 0x5U, 0, 0, 0x42A10388U
- #define **IOMUXC\_GPIO\_AD\_13\_LPUART3\_TX** 0x42A10140U, 0x6U, 0x42A10684U, 0x1U, 0x42A10388U
- #define **IOMUXC\_GPIO\_AD\_13\_USDHC2\_CD\_B** 0x42A10140U, 0x7U, 0x42A1092CU, 0x1-U, 0x42A10388U
- #define **IOMUXC\_GPIO\_AD\_13\_FLEXIO2\_FLEXIO13** 0x42A10140U, 0x8U, 0, 0, 0x42-A10388U
- #define **IOMUXC\_GPIO\_AD\_14\_SPDIF\_EXT\_CLK** 0x42A10144U, 0x0U, 0, 0, 0x42A1038C-U
- #define **IOMUXC\_GPIO\_AD\_14\_LPI2C5\_HREQ** 0x42A10144U, 0x1U, 0, 0, 0x42A1038CU
- #define **IOMUXC\_GPIO\_AD\_14\_GPT1\_COMPARE1** 0x42A10144U, 0x2U, 0, 0, 0x42A1038-CU
- #define **IOMUXC\_GPIO\_AD\_14\_KPP\_ROW06** 0x42A10144U, 0x3U, 0x42A105E0U, 0x1U,

- 0x42A1038CU
- #define **IOMUXC\_GPIO\_AD\_14\_FLEXPWM3\_PWMX00** 0x42A10144U, 0x4U, 0, 0, 0x42-A1038CU
  - #define **IOMUXC\_GPIO\_AD\_14\_GPIO4\_IO14** 0x42A10144U, 0x5U, 0, 0, 0x42A1038CU
  - #define **IOMUXC\_GPIO\_AD\_14\_LPUART3\_RX** 0x42A10144U, 0x6U, 0x42A10680U, 0x1U, 0x42A1038CU
  - #define **IOMUXC\_GPIO\_AD\_14\_USDHC2\_WP** 0x42A10144U, 0x7U, 0x42A10930U, 0x1U, 0x42A1038CU
  - #define **IOMUXC\_GPIO\_AD\_14\_FLEXIO2\_FLEXIO14** 0x42A10144U, 0x8U, 0, 0, 0x42-A1038CU
  - #define **IOMUXC\_GPIO\_AD\_15\_ECAT\_CLK\_ECAT\_CLK25** 0x42A10148U, 0x4CU, 0, 0, 0x42A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_SPDIF\_IN** 0x42A10148U, 0x0U, 0x42A10908U, 0x1U, 0x42-A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_LPUART10\_TX** 0x42A10148U, 0x1U, 0x42A10670U, 0x0U, 0x42A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_GPT1\_COMPARE2** 0x42A10148U, 0x2U, 0, 0, 0x42-A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_KPP\_COL06** 0x42A10148U, 0x3U, 0x42A105C0U, 0x1U, 0x42A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_FLEXPWM3\_PWMX01** 0x42A10148U, 0x4U, 0, 0, 0x42-A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_GPIO4\_IO15** 0x42A10148U, 0x5U, 0, 0, 0x42A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_LPUART3\_CTS\_B** 0x42A10148U, 0x6U, 0x42A1067CU, 0x1U, 0x42A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_LPSPI3\_PCS1** 0x42A10148U, 0x7U, 0x42A1060CU, 0x1U, 0x42A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_FLEXIO2\_FLEXIO15** 0x42A10148U, 0x8U, 0, 0, 0x42-A10390U
  - #define **IOMUXC\_GPIO\_AD\_15\_CAN1\_TX** 0x42A10148U, 0x9U, 0, 0, 0x42A10390U
  - #define **IOMUXC\_GPIO\_AD\_16\_SPDIF\_OUT** 0x42A1014CU, 0x0U, 0, 0, 0x42A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_LPUART10\_RX** 0x42A1014CU, 0x1U, 0x42A1066CU, 0x0U, 0x42A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_GPT1\_COMPARE3** 0x42A1014CU, 0x2U, 0, 0, 0x42-A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_KPP\_ROW05** 0x42A1014CU, 0x3U, 0x42A105DCU, 0x1U, 0x42A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_FLEXPWM3\_PWMX02** 0x42A1014CU, 0x4U, 0, 0, 0x42-A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_GPIO4\_IO16** 0x42A1014CU, 0x5U, 0, 0, 0x42A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_LPUART3\_RTS\_B** 0x42A1014CU, 0x6U, 0, 0, 0x42A10394-U
  - #define **IOMUXC\_GPIO\_AD\_16\_LPSPI3\_SCK** 0x42A1014CU, 0x7U, 0x42A10618U, 0x1U, 0x42A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_FLEXIO2\_FLEXIO16** 0x42A1014CU, 0x8U, 0, 0, 0x42-A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_CAN1\_RX** 0x42A1014CU, 0x9U, 0x42A104A0U, 0x0U, 0x42A10394U
  - #define **IOMUXC\_GPIO\_AD\_16\_ECAT\_LINK\_0** 0x42A1014CU, 0x4CU, 0, 0, 0x42A10394U
  - #define **IOMUXC\_GPIO\_AD\_17\_SAI4\_MCLK** 0x42A10150U, 0x0U, 0x42A108B0U, 0x0U,

- 0x42A10398U
- #define **IOMUXC\_GPIO\_AD\_17\_ACMP1\_CMPO** 0x42A10150U, 0x1U, 0, 0, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_GPT1\_CLK** 0x42A10150U, 0x2U, 0, 0, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_KPP\_COL05** 0x42A10150U, 0x3U, 0x42A105BCU, 0x1U, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_FLEXPWM3\_PWMX03** 0x42A10150U, 0x4U, 0, 0, 0x42-A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_GPIO4\_IO17** 0x42A10150U, 0x5U, 0, 0, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_I3C2\_PUR** 0x42A10150U, 0x6U, 0, 0, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_LPSPI3\_PCS0** 0x42A10150U, 0x7U, 0x42A10608U, 0x1U, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_FLEXIO2\_FLEXIO17** 0x42A10150U, 0x8U, 0, 0, 0x42-A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_LPI2C3\_HREQ** 0x42A10150U, 0x9U, 0, 0, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_17\_ECAT\_LINK\_1** 0x42A10150U, 0xCU, 0, 0, 0x42A10398U
  - #define **IOMUXC\_GPIO\_AD\_18\_ECAT\_PROM\_CLK** 0x42A10154U, 0xCU, 0, 0, 0x42-A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_SAI4\_RX\_SYNC** 0x42A10154U, 0x0U, 0x42A108C0U, 0x0-U, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_ACMP2\_CMPO** 0x42A10154U, 0x1U, 0, 0, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_LPUART5\_RI\_B** 0x42A10154U, 0x2U, 0x42A10698U, 0x0-U, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_KPP\_ROW04** 0x42A10154U, 0x3U, 0x42A105D8U, 0x1U, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_FLEXPWM4\_PWMX00** 0x42A10154U, 0x4U, 0, 0, 0x42-A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_GPIO4\_IO18** 0x42A10154U, 0x5U, 0, 0, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_I3C2\_SCL** 0x42A10154U, 0x6U, 0x42A105A0U, 0x0U, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_LPSPI3\_SDO** 0x42A10154U, 0x7U, 0x42A10620U, 0x1U, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_FLEXIO2\_FLEXIO18** 0x42A10154U, 0x8U, 0, 0, 0x42-A1039CU
  - #define **IOMUXC\_GPIO\_AD\_18\_LPI2C3\_SCL** 0x42A10154U, 0x9U, 0x42A105E8U, 0x2U, 0x42A1039CU
  - #define **IOMUXC\_GPIO\_AD\_19\_SAI4\_RX\_BCLK** 0x42A10158U, 0x0U, 0x42A108B4U, 0x0-U, 0x42A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_ACMP3\_CMPO** 0x42A10158U, 0x1U, 0, 0, 0x42A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_XBAR1\_XBAR\_INOUT19** 0x42A10158U, 0x2U, 0x42-A10944U, 0x0U, 0x42A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_KPP\_COL04** 0x42A10158U, 0x3U, 0x42A105B8U, 0x1U, 0x42A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_FLEXPWM4\_PWMX01** 0x42A10158U, 0x4U, 0, 0, 0x42-A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_GPIO4\_IO19** 0x42A10158U, 0x5U, 0, 0, 0x42A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_I3C2\_SDA** 0x42A10158U, 0x6U, 0x42A105A4U, 0x0U, 0x42A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_LPSPI3\_SDI** 0x42A10158U, 0x7U, 0x42A1061CU, 0x1U, 0x42A103A0U
  - #define **IOMUXC\_GPIO\_AD\_19\_FLEXIO2\_FLEXIO19** 0x42A10158U, 0x8U, 0, 0, 0x42-A103A0U

- #define **IOMUXC\_GPIO\_AD\_19\_LPI2C3\_SDA** 0x42A10158U, 0x9U, 0x42A105ECU, 0x2U, 0x42A103A0U
- #define **IOMUXC\_GPIO\_AD\_19\_ECAT\_PROM\_DATA** 0x42A10158U, 0xCU, 0x42A104F0-U, 0x0U, 0x42A103A0U
- #define **IOMUXC\_GPIO\_AD\_20\_SAI4\_RX\_DATA00** 0x42A1015CU, 0x0U, 0x42A108B8U, 0x0U, 0x42A103A4U
- #define **IOMUXC\_GPIO\_AD\_20\_ACMP4\_CMPO** 0x42A1015CU, 0x1U, 0, 0, 0x42A103A4U
- #define **IOMUXC\_GPIO\_AD\_20\_LPIT2\_TRIGGER00** 0x42A1015CU, 0x2U, 0, 0, 0x42A103-A4U
- #define **IOMUXC\_GPIO\_AD\_20\_SINC1\_EMCLK00** 0x42A1015CU, 0x3U, 0x42A108E4U, 0x1U, 0x42A103A4U
- #define **IOMUXC\_GPIO\_AD\_20\_FLEXPWM4\_PWMX02** 0x42A1015CU, 0x4U, 0, 0, 0x42-A103A4U
- #define **IOMUXC\_GPIO\_AD\_20\_GPIO4\_IO20** 0x42A1015CU, 0x5U, 0, 0, 0x42A103A4U
- #define **IOMUXC\_GPIO\_AD\_20\_NETC\_TMR\_TRIG1** 0x42A1015CU, 0x6U, 0x42A107CCU, 0x0U, 0x42A103A4U
- #define **IOMUXC\_GPIO\_AD\_20\_NETC\_1588\_CLK** 0x42A1015CU, 0x7U, 0x42A107D4U, 0x0U, 0x42A103A4U
- #define **IOMUXC\_GPIO\_AD\_20\_FLEXIO2\_FLEXIO20** 0x42A1015CU, 0x8U, 0, 0, 0x42-A103A4U
- #define **IOMUXC\_GPIO\_AD\_21\_SAI4\_TX\_DATA00** 0x42A10160U, 0x0U, 0, 0, 0x42A103-A8U
- #define **IOMUXC\_GPIO\_AD\_21\_LPIT2\_TRIGGER01** 0x42A10160U, 0x2U, 0, 0, 0x42A103-A8U
- #define **IOMUXC\_GPIO\_AD\_21\_SINC1\_EMBIT00** 0x42A10160U, 0x3U, 0x42A108D4U, 0x1U, 0x42A103A8U
- #define **IOMUXC\_GPIO\_AD\_21\_FLEXPWM4\_PWMX03** 0x42A10160U, 0x4U, 0, 0, 0x42-A103A8U
- #define **IOMUXC\_GPIO\_AD\_21\_GPIO4\_IO21** 0x42A10160U, 0x5U, 0, 0, 0x42A103A8U
- #define **IOMUXC\_GPIO\_AD\_21\_NETC\_TMR\_TRIG2** 0x42A10160U, 0x6U, 0x42A107D0U, 0x0U, 0x42A103A8U
- #define **IOMUXC\_GPIO\_AD\_21\_NETC\_TMR\_GCLK** 0x42A10160U, 0x7U, 0, 0, 0x42A103-A8U
- #define **IOMUXC\_GPIO\_AD\_21\_FLEXIO2\_FLEXIO21** 0x42A10160U, 0x8U, 0, 0, 0x42-A103A8U
- #define **IOMUXC\_GPIO\_AD\_21\_ECAT\_LED\_RUN** 0x42A10160U, 0xCU, 0, 0, 0x42A103A8-U
- #define **IOMUXC\_GPIO\_AD\_22\_ECAT\_LED\_ERR** 0x42A10164U, 0xCU, 0, 0, 0x42A103A-CU
- #define **IOMUXC\_GPIO\_AD\_22\_SAI4\_TX\_BCLK** 0x42A10164U, 0x0U, 0x42A108C4U, 0x0-U, 0x42A103ACU
- #define **IOMUXC\_GPIO\_AD\_22\_LPIT2\_TRIGGER02** 0x42A10164U, 0x2U, 0, 0, 0x42A103-ACU
- #define **IOMUXC\_GPIO\_AD\_22\_SINC1\_EMCLK01** 0x42A10164U, 0x3U, 0x42A108E8U, 0x1U, 0x42A103ACU
- #define **IOMUXC\_GPIO\_AD\_22\_GPIO4\_IO22** 0x42A10164U, 0x5U, 0, 0, 0x42A103ACU
- #define **IOMUXC\_GPIO\_AD\_22\_NETC\_TMR\_ALARM1** 0x42A10164U, 0x7U, 0, 0, 0x42-A103ACU
- #define **IOMUXC\_GPIO\_AD\_22\_FLEXIO2\_FLEXIO22** 0x42A10164U, 0x8U, 0, 0, 0x42-A103ACU

- #define **IOMUXC\_GPIO\_AD\_23\_SAI4\_TX\_SYNC** 0x42A10168U, 0x0U, 0x42A108C8U, 0x0U, 0x42A103B0U
- #define **IOMUXC\_GPIO\_AD\_23\_LPIT2\_TRIGGER03** 0x42A10168U, 0x2U, 0, 0, 0x42A103B0U
- #define **IOMUXC\_GPIO\_AD\_23\_SINC1\_EMBIT01** 0x42A10168U, 0x3U, 0x42A108D8U, 0x1U, 0x42A103B0U
- #define **IOMUXC\_GPIO\_AD\_23\_GPIO4\_IO23** 0x42A10168U, 0x5U, 0, 0, 0x42A103B0U
- #define **IOMUXC\_GPIO\_AD\_23\_NETC\_TMR\_ALARM2** 0x42A10168U, 0x7U, 0, 0, 0x42A103B0U
- #define **IOMUXC\_GPIO\_AD\_23\_FLEXIO2\_FLEXIO23** 0x42A10168U, 0x8U, 0, 0, 0x42A103B0U
- #define **IOMUXC\_GPIO\_AD\_23\_ECAT\_LED\_STATE\_RUN** 0x42A10168U, 0x9U, 0, 0, 0x42A103B0U
- #define **IOMUXC\_GPIO\_AD\_24\_LPUART6\_TX** 0x42A1016CU, 0x0U, 0x42A106B8U, 0x1U, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_24\_LPI2C4\_SCL** 0x42A1016CU, 0x1U, 0x42A105F0U, 0x0U, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_24\_SINC2\_MOD\_CLK1** 0x42A1016CU, 0x3U, 0, 0, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_24\_FLEXPWM2\_PWMA00** 0x42A1016CU, 0x4U, 0x42A1050CU, 0x1U, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_24\_GPIO4\_IO24** 0x42A1016CU, 0x5U, 0, 0, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_24\_NETC\_TMR\_TRIG1** 0x42A1016CU, 0x7U, 0x42A107CCU, 0x1U, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_24\_FLEXIO2\_FLEXIO24** 0x42A1016CU, 0x8U, 0, 0, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_24\_ECAT\_LINK\_ACT00** 0x42A1016CU, 0x9U, 0, 0, 0x42A103B4U
- #define **IOMUXC\_GPIO\_AD\_25\_ECAT\_LINK\_ACT01** 0x42A10170U, 0x9U, 0, 0, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_LPUART6\_RX** 0x42A10170U, 0x0U, 0x42A106B4U, 0x1U, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_LPI2C4\_SDA** 0x42A10170U, 0x1U, 0x42A105F4U, 0x0U, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_LPSPi5\_PCS3** 0x42A10170U, 0x2U, 0x42A10640U, 0x1U, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_SINC2\_MOD\_CLK2** 0x42A10170U, 0x3U, 0, 0, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_FLEXPWM2\_PWMB00** 0x42A10170U, 0x4U, 0x42A10518U, 0x1U, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_GPIO4\_IO25** 0x42A10170U, 0x5U, 0, 0, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_NETC\_TMR\_TRIG2** 0x42A10170U, 0x7U, 0x42A107D0U, 0x1U, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_25\_FLEXIO2\_FLEXIO25** 0x42A10170U, 0x8U, 0, 0, 0x42A103B8U
- #define **IOMUXC\_GPIO\_AD\_26\_LPUART6\_CTS\_B** 0x42A10174U, 0x0U, 0x42A106A4U, 0x1U, 0x42A103BCU
- #define **IOMUXC\_GPIO\_AD\_26\_LPUART5\_TX** 0x42A10174U, 0x1U, 0x42A106A0U, 0x3U, 0x42A103BCU
- #define **IOMUXC\_GPIO\_AD\_26\_LPSPi5\_PCS2** 0x42A10174U, 0x2U, 0x42A1063CU, 0x1U,

- 0x42A103BCU
- #define **IOMUXC\_GPIO\_AD\_26\_SINC2\_EMCLK00** 0x42A10174U, 0x3U, 0x42A108FCU, 0x0U, 0x42A103BCU
  - #define **IOMUXC\_GPIO\_AD\_26\_FLEXPWM2\_PWMA01** 0x42A10174U, 0x4U, 0x42A10510-U, 0x1U, 0x42A103BCU
  - #define **IOMUXC\_GPIO\_AD\_26\_GPIO4\_IO26** 0x42A10174U, 0x5U, 0, 0, 0x42A103BCU
  - #define **IOMUXC\_GPIO\_AD\_26\_KPP\_ROW00** 0x42A10174U, 0x6U, 0x42A105C8U, 0x1U, 0x42A103BCU
  - #define **IOMUXC\_GPIO\_AD\_26\_NETC\_TMR\_PP1** 0x42A10174U, 0x7U, 0, 0, 0x42A103BCU
  - #define **IOMUXC\_GPIO\_AD\_26\_FLEXIO2\_FLEXIO26** 0x42A10174U, 0x8U, 0, 0, 0x42-A103BCU
  - #define **IOMUXC\_GPIO\_AD\_26\_USDHC2\_CD\_B** 0x42A10174U, 0x9U, 0x42A1092CU, 0x2-U, 0x42A103BCU
  - #define **IOMUXC\_GPIO\_AD\_26\_MIC\_BITSTREAM02** 0x42A10174U, 0xCU, 0x42A106D8-U, 0x1U, 0x42A103BCU
  - #define **IOMUXC\_GPIO\_AD\_27\_LPUART6\_RTS\_B** 0x42A10178U, 0x0U, 0, 0, 0x42A103C0-U
  - #define **IOMUXC\_GPIO\_AD\_27\_LPUART5\_RX** 0x42A10178U, 0x1U, 0x42A1069CU, 0x3U, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_LPSPi5\_PCS1** 0x42A10178U, 0x2U, 0x42A10638U, 0x2U, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_SINC2\_EMBIT00** 0x42A10178U, 0x3U, 0, 0, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_FLEXPWM2\_PWMB01** 0x42A10178U, 0x4U, 0x42A1051-CU, 0x1U, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_GPIO4\_IO27** 0x42A10178U, 0x5U, 0, 0, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_KPP\_COL00** 0x42A10178U, 0x6U, 0x42A105A8U, 0x1U, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_NETC\_TMR\_PP2** 0x42A10178U, 0x7U, 0, 0, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_FLEXIO2\_FLEXIO27** 0x42A10178U, 0x8U, 0, 0, 0x42-A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_USDHC2\_WP** 0x42A10178U, 0x9U, 0x42A10930U, 0x2U, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_27\_MIC\_CLK** 0x42A10178U, 0xCU, 0, 0, 0x42A103C0U
  - #define **IOMUXC\_GPIO\_AD\_28\_MIC\_BITSTREAM00** 0x42A1017CU, 0xCU, 0x42A106D0-U, 0x1U, 0x42A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_LPSPi5\_SCK** 0x42A1017CU, 0x0U, 0x42A10644U, 0x2U, 0x42A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_I3C1\_PUR** 0x42A1017CU, 0x2U, 0, 0, 0x42A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_SINC2\_EMCLK01** 0x42A1017CU, 0x3U, 0, 0, 0x42A103C4-U
  - #define **IOMUXC\_GPIO\_AD\_28\_FLEXPWM2\_PWMB02** 0x42A1017CU, 0x4U, 0x42-A10520U, 0x1U, 0x42A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_GPIO4\_IO28** 0x42A1017CU, 0x5U, 0, 0, 0x42A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_KPP\_ROW03** 0x42A1017CU, 0x6U, 0x42A105D4U, 0x1U, 0x42A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_NETC\_TMR\_PP3** 0x42A1017CU, 0x7U, 0, 0, 0x42A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_FLEXIO2\_FLEXIO28** 0x42A1017CU, 0x8U, 0, 0, 0x42-A103C4U
  - #define **IOMUXC\_GPIO\_AD\_28\_USDHC2\_RESET\_B** 0x42A1017CU, 0x9U, 0, 0, 0x42A103-C4U
  - #define **IOMUXC\_GPIO\_AD\_29\_LPSPi5\_PCS0** 0x42A10180U, 0x0U, 0x42A10634U, 0x2U,

- 0x42A103C8U
- #define **IOMUXC\_GPIO\_AD\_29\_USDHC2\_CD\_B** 0x42A10180U, 0x2U, 0x42A1092CU, 0x3-U, 0x42A103C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_SINC2\_EMBIT01** 0x42A10180U, 0x3U, 0, 0, 0x42A103C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_FLEXPWM2\_PWMA02** 0x42A10180U, 0x4U, 0x42A10514-U, 0x1U, 0x42A103C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_GPIO4\_IO29** 0x42A10180U, 0x5U, 0, 0, 0x42A103C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_KPP\_COL03** 0x42A10180U, 0x6U, 0x42A105B4U, 0x1U, 0x42A103C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_EWM\_EWM\_OUT\_B** 0x42A10180U, 0x7U, 0, 0, 0x42A103-C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_FLEXIO2\_FLEXIO29** 0x42A10180U, 0x8U, 0, 0, 0x42-A103C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_USDHC2\_VSELECT** 0x42A10180U, 0x9U, 0, 0, 0x42A103-C8U
  - #define **IOMUXC\_GPIO\_AD\_29\_MIC\_BITSTREAM01** 0x42A10180U, 0xCU, 0x42A106D4-U, 0x1U, 0x42A103C8U
  - #define **IOMUXC\_GPIO\_AD\_30\_LPSPi5\_SDO** 0x42A10184U, 0x0U, 0x42A1064CU, 0x2U, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_USB\_OTG2\_OC** 0x42A10184U, 0x1U, 0x42A10914U, 0x1U, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_CAN2\_TX** 0x42A10184U, 0x2U, 0, 0, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_SINC2\_EMCLK02** 0x42A10184U, 0x3U, 0x42A10900U, 0x0U, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_LPUART8\_TX** 0x42A10184U, 0x4U, 0x42A106C4U, 0x0U, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_GPIO4\_IO30** 0x42A10184U, 0x5U, 0, 0, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_KPP\_ROW02** 0x42A10184U, 0x6U, 0x42A105D0U, 0x1U, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_NETC\_EMDC** 0x42A10184U, 0x7U, 0, 0, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_FLEXIO2\_FLEXIO30** 0x42A10184U, 0x8U, 0, 0, 0x42-A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_XBAR1\_XBAR\_INOUT24** 0x42A10184U, 0x9U, 0x42-A10958U, 0x1U, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_30\_ECAT\_MCLK** 0x42A10184U, 0xCU, 0, 0, 0x42A103CCU
  - #define **IOMUXC\_GPIO\_AD\_31\_LPSPi5\_SDI** 0x42A10188U, 0x0U, 0x42A10648U, 0x2U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_USB\_OTG2\_PWR** 0x42A10188U, 0x1U, 0, 0, 0x42A103D0-U
  - #define **IOMUXC\_GPIO\_AD\_31\_CAN2\_RX** 0x42A10188U, 0x2U, 0x42A104A4U, 0x1U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_SINC2\_EMBIT02** 0x42A10188U, 0x3U, 0x42A108F4U, 0x0-U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_LPUART8\_RX** 0x42A10188U, 0x4U, 0x42A106C0U, 0x0U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_GPIO4\_IO31** 0x42A10188U, 0x5U, 0, 0, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_KPP\_COL02** 0x42A10188U, 0x6U, 0x42A105B0U, 0x1U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_NETC\_EMDIO** 0x42A10188U, 0x7U, 0x42A10798U, 0x4U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_FLEXIO2\_FLEXIO31** 0x42A10188U, 0x8U, 0, 0, 0x42-

- A103D0U
- #define **IOMUXC\_GPIO\_AD\_31\_XBAR1\_XBAR\_INOUT25** 0x42A10188U, 0x9U, 0x42-A1095CU, 0x1U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_31\_ECAT\_MDIO** 0x42A10188U, 0xCU, 0x42A104ECU, 0x0U, 0x42A103D0U
  - #define **IOMUXC\_GPIO\_AD\_32\_MIC\_BITSTREAM03** 0x42A1018CU, 0xCU, 0x42A106DC-U, 0x1U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_LPI2C5\_SCL** 0x42A1018CU, 0x0U, 0x42A105F8U, 0x1U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_USBPHY2\_OTG\_ID** 0x42A1018CU, 0x1U, 0x42A10920U, 0x1U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_GPC\_PMIC\_RDY** 0x42A1018CU, 0x2U, 0, 0, 0x42A103D4-U
  - #define **IOMUXC\_GPIO\_AD\_32\_SINC2\_EMCLK03** 0x42A1018CU, 0x3U, 0x42A10904U, 0x0U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_USDHC1\_CD\_B** 0x42A1018CU, 0x4U, 0x42A10924U, 0x0-U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_GPIO5\_IO00** 0x42A1018CU, 0x5U, 0, 0, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_KPP\_ROW01** 0x42A1018CU, 0x6U, 0x42A105CCU, 0x1U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_NETC\_TMR\_TRIG1** 0x42A1018CU, 0x7U, 0x42A107CCU, 0x2U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_32\_LPUART10\_TX** 0x42A1018CU, 0x8U, 0x42A10670U, 0x1U, 0x42A103D4U
  - #define **IOMUXC\_GPIO\_AD\_33\_LPI2C5\_SDA** 0x42A10190U, 0x0U, 0x42A105FCU, 0x1U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_USBPHY1\_OTG\_ID** 0x42A10190U, 0x1U, 0x42A1091CU, 0x1U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_XBAR1\_XBAR\_INOUT17** 0x42A10190U, 0x2U, 0x42-A1093CU, 0x0U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_SINC2\_EMBIT03** 0x42A10190U, 0x3U, 0x42A108F8U, 0x0-U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_USDHC1\_WP** 0x42A10190U, 0x4U, 0x42A10928U, 0x0U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_GPIO5\_IO01** 0x42A10190U, 0x5U, 0, 0, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_KPP\_COL01** 0x42A10190U, 0x6U, 0x42A105ACU, 0x1U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_NETC\_TMR\_TRIG2** 0x42A10190U, 0x7U, 0x42A107D0U, 0x2U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_33\_LPUART10\_RX** 0x42A10190U, 0x8U, 0x42A1066CU, 0x1-U, 0x42A103D8U
  - #define **IOMUXC\_GPIO\_AD\_34\_I3C2\_SCL** 0x42A10194U, 0x0U, 0x42A105A0U, 0x1U, 0x42A103DCU
  - #define **IOMUXC\_GPIO\_AD\_34\_USB\_OTG1\_PWR** 0x42A10194U, 0x1U, 0, 0, 0x42A103DC-U
  - #define **IOMUXC\_GPIO\_AD\_34\_XBAR1\_XBAR\_INOUT18** 0x42A10194U, 0x2U, 0x42-A10940U, 0x1U, 0x42A103DCU
  - #define **IOMUXC\_GPIO\_AD\_34\_SINC\_FILTER\_GLUE2\_BREAK** 0x42A10194U, 0x3U, 0, 0, 0x42A103DCU
  - #define **IOMUXC\_GPIO\_AD\_34\_USDHC1\_VSELECT** 0x42A10194U, 0x4U, 0, 0, 0x42A103-

DCU

- #define **IOMUXC\_GPIO\_AD\_34\_GPIO5\_IO02** 0x42A10194U, 0x5U, 0, 0, 0x42A103DCU
- #define **IOMUXC\_GPIO\_AD\_34\_NETC\_TMR\_ALARM1** 0x42A10194U, 0x7U, 0, 0, 0x42A103DCU
- #define **IOMUXC\_GPIO\_AD\_34\_LPUART10\_CTS\_B** 0x42A10194U, 0x8U, 0, 0, 0x42A103DCU
- #define **IOMUXC\_GPIO\_AD\_35\_I3C2\_SDA** 0x42A10198U, 0x0U, 0x42A105A4U, 0x1U, 0x42A103E0U
- #define **IOMUXC\_GPIO\_AD\_35\_USB\_OTG1\_OC** 0x42A10198U, 0x1U, 0x42A10918U, 0x1U, 0x42A103E0U
- #define **IOMUXC\_GPIO\_AD\_35\_XBAR1\_XBAR\_INOUT19** 0x42A10198U, 0x2U, 0x42A10944U, 0x1U, 0x42A103E0U
- #define **IOMUXC\_GPIO\_AD\_35\_SINC2\_MOD\_CLK0** 0x42A10198U, 0x3U, 0, 0, 0x42A103E0U
- #define **IOMUXC\_GPIO\_AD\_35\_USDHC1\_RESET\_B** 0x42A10198U, 0x4U, 0, 0, 0x42A103E0U
- #define **IOMUXC\_GPIO\_AD\_35\_GPIO5\_IO03** 0x42A10198U, 0x5U, 0, 0, 0x42A103E0U
- #define **IOMUXC\_GPIO\_AD\_35\_NETC\_TMR\_ALARM2** 0x42A10198U, 0x7U, 0, 0, 0x42A103E0U
- #define **IOMUXC\_GPIO\_AD\_35\_LPUART10\_RTS\_B** 0x42A10198U, 0x8U, 0, 0, 0x42A103E0U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_USDHC1\_CMD** 0x42A1019CU, 0x0U, 0, 0, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_SINC1\_EMCLK02** 0x42A1019CU, 0x1U, 0x42A108ECU, 0x1U, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_XBAR1\_XBAR\_INOUT20** 0x42A1019CU, 0x2U, 0x42A10948U, 0x1U, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_LPTMR2\_ALT1** 0x42A1019CU, 0x3U, 0, 0, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_XSPI\_SLV\_CS** 0x42A1019CU, 0x4U, 0x42A10A00U, 0x0U, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_GPIO5\_IO04** 0x42A1019CU, 0x5U, 0, 0, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_LPSPi3\_PCS0** 0x42A1019CU, 0x6U, 0x42A10608U, 0x2U, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_KPP\_ROW07** 0x42A1019CU, 0x8U, 0x42A105E4U, 0x2U, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_00\_CCM\_CLK01** 0x42A1019CU, 0x0CU, 0, 0, 0x42A103E4U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_USDHC1\_CLK** 0x42A101A0U, 0x0U, 0, 0, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_SINC1\_EMBIT02** 0x42A101A0U, 0x1U, 0x42A108DCU, 0x1U, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_XBAR1\_XBAR\_INOUT21** 0x42A101A0U, 0x2U, 0x42A1094CU, 0x1U, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_LPTMR2\_ALT2** 0x42A101A0U, 0x3U, 0, 0, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_XSPI\_SLV\_CLK** 0x42A101A0U, 0x4U, 0x42A10A28U, 0x0U, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_GPIO5\_IO05** 0x42A101A0U, 0x5U, 0, 0, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_LPSPi3\_SCK** 0x42A101A0U, 0x6U, 0x42A10618U, 0x2U

- U, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_KPP\_COL07** 0x42A101A0U, 0x8U, 0x42A105C4U, 0x2-U, 0x42A103E8U
- #define **IOMUXC\_GPIO\_SD\_B1\_01\_CCM\_CLKO2** 0x42A101A0U, 0xCU, 0, 0, 0x42A103E8-U
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_ECAT\_RESET\_OUT** 0x42A101A4U, 0xCU, 0, 0, 0x42-A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_USDHC1\_DATA0** 0x42A101A4U, 0x0U, 0, 0, 0x42A103-ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_SINC1\_EMCLK03** 0x42A101A4U, 0x1U, 0x42A108F0-U, 0x1U, 0x42A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_XBAR1\_XBAR\_INOUT22** 0x42A101A4U, 0x2U, 0x42-A10950U, 0x1U, 0x42A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_LPTMR2\_ALT3** 0x42A101A4U, 0x3U, 0, 0, 0x42A103-ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_XSPI\_SLV\_DATA04** 0x42A101A4U, 0x4U, 0x42A10-A18U, 0x0U, 0x42A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_GPIO5\_IO06** 0x42A101A4U, 0x5U, 0, 0, 0x42A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_LPSPi3\_SDO** 0x42A101A4U, 0x6U, 0x42A10620U, 0x2-U, 0x42A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_KPP\_ROW06** 0x42A101A4U, 0x8U, 0x42A105E0U, 0x2-U, 0x42A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_02\_FLEXSPI1\_BUS2BIT\_A\_SS1\_B** 0x42A101A4U, 0x9U, 0, 0, 0x42A103ECU
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_USDHC1\_DATA1** 0x42A101A8U, 0x0U, 0, 0, 0x42A103-F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_SINC1\_EMBIT03** 0x42A101A8U, 0x1U, 0x42A108E0U, 0x1U, 0x42A103F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_XBAR1\_XBAR\_INOUT23** 0x42A101A8U, 0x2U, 0x42-A10954U, 0x1U, 0x42A103F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_LPTMR3\_ALT1** 0x42A101A8U, 0x3U, 0, 0, 0x42A103-F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_XSPI\_SLV\_DATA05** 0x42A101A8U, 0x4U, 0x42A10A1-CU, 0x0U, 0x42A103F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_GPIO5\_IO07** 0x42A101A8U, 0x5U, 0, 0, 0x42A103F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_LPSPi3\_SDI** 0x42A101A8U, 0x6U, 0x42A1061CU, 0x2-U, 0x42A103F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_KPP\_COL06** 0x42A101A8U, 0x8U, 0x42A105C0U, 0x2-U, 0x42A103F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_03\_FLEXSPI1\_BUS2BIT\_B\_SS1\_B** 0x42A101A8U, 0x9U, 0, 0, 0x42A103F0U
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_USDHC1\_DATA2** 0x42A101ACU, 0x0U, 0, 0, 0x42-A103F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_SINC\_FILTER\_GLUE1\_BREAK** 0x42A101ACU, 0x1-U, 0, 0, 0x42A103F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_SINC2\_EMCLK02** 0x42A101ACU, 0x2U, 0x42A10900-U, 0x1U, 0x42A103F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_LPTMR3\_ALT2** 0x42A101ACU, 0x3U, 0, 0, 0x42A103-F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_XSPI\_SLV\_DATA06** 0x42A101ACU, 0x4U, 0x42A10-

- A20U, 0x0U, 0x42A103F4U
- #define **IOMUXC\_GPIO\_SD\_B1\_04\_GPIO5\_IO08** 0x42A101ACU, 0x5U, 0, 0, 0x42A103F4U
  - #define **IOMUXC\_GPIO\_SD\_B1\_04\_LPSPi3\_PCS1** 0x42A101ACU, 0x6U, 0x42A1060CU, 0x2U, 0x42A103F4U
  - #define **IOMUXC\_GPIO\_SD\_B1\_04\_FLEXSPI1\_BUS2BIT\_B\_SS0\_B** 0x42A101ACU, 0x8U, 0, 0, 0x42A103F4U
  - #define **IOMUXC\_GPIO\_SD\_B1\_04\_FLEXSPI1\_BUS2BIT\_A\_SS1\_B** 0x42A101ACU, 0x9U, 0, 0, 0x42A103F4U
  - #define **IOMUXC\_GPIO\_SD\_B1\_05\_USDHC1\_DATA3** 0x42A101B0U, 0x0U, 0, 0, 0x42A103F8U
  - #define **IOMUXC\_GPIO\_SD\_B1\_05\_SINC2\_EMBIT02** 0x42A101B0U, 0x2U, 0x42A108F4U, 0x1U, 0x42A103F8U
  - #define **IOMUXC\_GPIO\_SD\_B1\_05\_LPTMR3\_ALT3** 0x42A101B0U, 0x3U, 0, 0, 0x42A103F8U
  - #define **IOMUXC\_GPIO\_SD\_B1\_05\_XSPI\_SLV\_DATA07** 0x42A101B0U, 0x4U, 0x42A10A24U, 0x0U, 0x42A103F8U
  - #define **IOMUXC\_GPIO\_SD\_B1\_05\_GPIO5\_IO09** 0x42A101B0U, 0x5U, 0, 0, 0x42A103F8U
  - #define **IOMUXC\_GPIO\_SD\_B1\_05\_LPSPi3\_PCS2** 0x42A101B0U, 0x6U, 0x42A10610U, 0x1U, 0x42A103F8U
  - #define **IOMUXC\_GPIO\_SD\_B1\_05\_FLEXSPI1\_BUS2BIT\_B\_SS0\_B** 0x42A101B0U, 0x9U, 0, 0, 0x42A103F8U
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_MIC\_BITSTREAM00** 0x42A101B4U, 0xCU, 0x42A106D0U, 0x2U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_USDHC2\_DATA3** 0x42A101B4U, 0x0U, 0, 0, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_FLEXSPI1\_BUS2BIT\_B\_DATA04** 0x42A101B4U, 0x1U, 0x42A1055CU, 0x0U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_XSPI\_SLV\_DATA04** 0x42A101B4U, 0x2U, 0x42A10A18U, 0x1U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_XBAR1\_XBAR\_INOUT17** 0x42A101B4U, 0x3U, 0x42A1093CU, 0x1U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_KPP\_ROW01** 0x42A101B4U, 0x4U, 0x42A105CCU, 0x2U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_GPIO5\_IO10** 0x42A101B4U, 0x5U, 0, 0, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_LPSPi3\_PCS3** 0x42A101B4U, 0x6U, 0x42A10614U, 0x1U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_NETC\_1588\_CLK** 0x42A101B4U, 0x8U, 0x42A107D4U, 0x1U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_00\_LPUART8\_TX** 0x42A101B4U, 0x9U, 0x42A106C4U, 0x1U, 0x42A103FCU
  - #define **IOMUXC\_GPIO\_SD\_B2\_01\_USDHC2\_DATA2** 0x42A101B8U, 0x0U, 0, 0, 0x42A10400U
  - #define **IOMUXC\_GPIO\_SD\_B2\_01\_FLEXSPI1\_BUS2BIT\_B\_DATA05** 0x42A101B8U, 0x1U, 0x42A10560U, 0x0U, 0x42A10400U
  - #define **IOMUXC\_GPIO\_SD\_B2\_01\_XSPI\_SLV\_DATA05** 0x42A101B8U, 0x2U, 0x42A10A1CU, 0x1U, 0x42A10400U
  - #define **IOMUXC\_GPIO\_SD\_B2\_01\_QTIMER6\_TIMER0** 0x42A101B8U, 0x3U, 0x42A10894U, 0x2U, 0x42A10400U
  - #define **IOMUXC\_GPIO\_SD\_B2\_01\_KPP\_COL01** 0x42A101B8U, 0x4U, 0x42A105ACU, 0x2U, 0x42A10400U

- #define **IOMUXC\_GPIO\_SD\_B2\_01\_GPIO5\_IO11** 0x42A101B8U, 0x5U, 0, 0, 0x42A10400U
- #define **IOMUXC\_GPIO\_SD\_B2\_01\_NETC\_TMR\_GCLK** 0x42A101B8U, 0x8U, 0, 0, 0x42A10400U
- #define **IOMUXC\_GPIO\_SD\_B2\_01\_LPUART8\_RX** 0x42A101B8U, 0x9U, 0x42A106C0U, 0x1U, 0x42A10400U
- #define **IOMUXC\_GPIO\_SD\_B2\_01\_MIC\_BITSTREAM01** 0x42A101B8U, 0xCU, 0x42A106D4U, 0x2U, 0x42A10400U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_MIC\_BITSTREAM02** 0x42A101BCU, 0xCU, 0x42A106D8U, 0x2U, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_USDHC2\_DATA1** 0x42A101BCU, 0x0U, 0, 0, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_FLEXSPI1\_BUS2BIT\_B\_DATA06** 0x42A101BCU, 0x1U, 0x42A10564U, 0x0U, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_XSPI\_SLV\_DATA06** 0x42A101BCU, 0x2U, 0x42A10A20U, 0x1U, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_QTIMER6\_TIMER1** 0x42A101BCU, 0x3U, 0x42A10898U, 0x2U, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_KPP\_ROW00** 0x42A101BCU, 0x4U, 0x42A105C8U, 0x2U, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_GPIO5\_IO12** 0x42A101BCU, 0x5U, 0, 0, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_NETC\_TMR\_ALARM1** 0x42A101BCU, 0x8U, 0, 0, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_02\_LPUART8\_CTS\_B** 0x42A101BCU, 0x9U, 0x42A106BCU, 0x0U, 0x42A10404U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_MIC\_BITSTREAM03** 0x42A101C0U, 0xCU, 0x42A106DCU, 0x2U, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_USDHC2\_DATA0** 0x42A101C0U, 0x0U, 0, 0, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_FLEXSPI1\_BUS2BIT\_B\_DATA07** 0x42A101C0U, 0x1U, 0x42A10568U, 0x0U, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_XSPI\_SLV\_DATA07** 0x42A101C0U, 0x2U, 0x42A10A24U, 0x1U, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_QTIMER6\_TIMER2** 0x42A101C0U, 0x3U, 0x42A1089CU, 0x1U, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_KPP\_COL00** 0x42A101C0U, 0x4U, 0x42A105A8U, 0x2U, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_GPIO5\_IO13** 0x42A101C0U, 0x5U, 0, 0, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_NETC\_TMR\_ALARM2** 0x42A101C0U, 0x8U, 0, 0, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_03\_LPUART8\_RTS\_B** 0x42A101C0U, 0x9U, 0, 0, 0x42A10408U
- #define **IOMUXC\_GPIO\_SD\_B2\_04\_USDHC2\_CLK** 0x42A101C4U, 0x0U, 0, 0, 0x42A1040CU
- #define **IOMUXC\_GPIO\_SD\_B2\_04\_FLEXSPI1\_BUS2BIT\_B\_SS1\_B** 0x42A101C4U, 0x1U, 0, 0, 0x42A1040CU
- #define **IOMUXC\_GPIO\_SD\_B2\_04\_QTIMER7\_TIMER0** 0x42A101C4U, 0x3U, 0x42A108A0U, 0x1U, 0x42A1040CU
- #define **IOMUXC\_GPIO\_SD\_B2\_04\_KPP\_ROW03** 0x42A101C4U, 0x4U, 0x42A105D4U, 0x2U, 0x42A1040CU
- #define **IOMUXC\_GPIO\_SD\_B2\_04\_GPIO5\_IO14** 0x42A101C4U, 0x5U, 0, 0, 0x42A1040CU

- #define **IOMUXC\_GPIO\_SD\_B2\_04\_LPUART5\_RI\_B** 0x42A101C4U, 0x6U, 0x42A10698U, 0x1U, 0x42A1040CU
- #define **IOMUXC\_GPIO\_SD\_B2\_04\_NETC\_TMR\_PP1** 0x42A101C4U, 0x8U, 0, 0, 0x42-A1040CU
- #define **IOMUXC\_GPIO\_SD\_B2\_04\_MIC\_CLK** 0x42A101C4U, 0xCU, 0, 0, 0x42A1040CU
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_USDHC2\_CMD** 0x42A101C8U, 0x0U, 0, 0, 0x42A10410-U
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_FLEXSPI1\_BUS2BIT\_B\_DQS** 0x42A101C8U, 0x1U, 0x42A10548U, 0x0U, 0x42A10410U
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_XSPI\_SLV\_DQS** 0x42A101C8U, 0x2U, 0x42A10A04U, 0x0U, 0x42A10410U
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_QTIMER7\_TIMER1** 0x42A101C8U, 0x3U, 0x42A108-A4U, 0x1U, 0x42A10410U
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_LPSPi4\_PCS3** 0x42A101C8U, 0x4U, 0, 0, 0x42A10410U
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_GPIO5\_IO15** 0x42A101C8U, 0x5U, 0, 0, 0x42A10410U
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_LPUART5\_DTR\_B** 0x42A101C8U, 0x6U, 0, 0, 0x42-A10410U
- #define **IOMUXC\_GPIO\_SD\_B2\_05\_NETC\_TMR\_PP2** 0x42A101C8U, 0x8U, 0, 0, 0x42-A10410U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_USDHC2\_RESET\_B** 0x42A101CCU, 0x0U, 0, 0, 0x42-A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_FLEXSPI1\_BUS2BIT\_B\_SS0\_B** 0x42A101CCU, 0x1U, 0, 0, 0x42A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_XSPI\_SLV\_CS** 0x42A101CCU, 0x2U, 0x42A10A00U, 0x1U, 0x42A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_QTIMER7\_TIMER2** 0x42A101CCU, 0x3U, 0, 0, 0x42-A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_LPSPi4\_PCS2** 0x42A101CCU, 0x4U, 0, 0, 0x42A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_GPIO5\_IO16** 0x42A101CCU, 0x5U, 0, 0, 0x42A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_LPUART5\_CTS\_B** 0x42A101CCU, 0x6U, 0x42A1068C-U, 0x2U, 0x42A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_06\_NETC\_TMR\_PP3** 0x42A101CCU, 0x8U, 0, 0, 0x42-A10414U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_USDHC2\_STROBE** 0x42A101D0U, 0x0U, 0, 0, 0x42-A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_FLEXSPI1\_BUS2BIT\_B\_SCLK** 0x42A101D0U, 0x1U, 0x42A1056CU, 0x0U, 0x42A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_XSPI\_SLV\_CLK** 0x42A101D0U, 0x2U, 0x42A10A28U, 0x1U, 0x42A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_QTIMER7\_TIMER3** 0x42A101D0U, 0x3U, 0, 0, 0x42-A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_LPSPi4\_PCS1** 0x42A101D0U, 0x4U, 0, 0, 0x42A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_GPIO5\_IO17** 0x42A101D0U, 0x5U, 0, 0, 0x42A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_LPUART5 RTS\_B** 0x42A101D0U, 0x6U, 0, 0, 0x42-A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_07\_NETC\_TMR\_ALARM1** 0x42A101D0U, 0x8U, 0, 0, 0x42A10418U
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_USDHC2\_DATA4** 0x42A101D4U, 0x0U, 0, 0, 0x42-A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_FLEXSPI1\_BUS2BIT\_B\_DATA00** 0x42A101D4U, 0x1-

- U, 0x42A1054CU, 0x0U, 0x42A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_XSPI\_SLV\_DATA00** 0x42A101D4U, 0x2U, 0x42A10-A08U, 0x0U, 0x42A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_QTIMER8\_TIMER0** 0x42A101D4U, 0x3U, 0x42A108-A8U, 0x1U, 0x42A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_LPSPi4\_SCK** 0x42A101D4U, 0x4U, 0x42A10628U, 0x1-U, 0x42A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_GPIO5\_IO18** 0x42A101D4U, 0x5U, 0, 0, 0x42A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_LPUART5\_TX** 0x42A101D4U, 0x6U, 0x42A106A0U, 0x4U, 0x42A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_NETC\_TMR\_ALARM2** 0x42A101D4U, 0x8U, 0, 0, 0x42A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_08\_NETC\_TMR\_PP2** 0x42A101D4U, 0x9U, 0, 0, 0x42-A1041CU
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_USDHC2\_DATA5** 0x42A101D8U, 0x0U, 0, 0, 0x42-A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_FLEXSPi1\_BUS2BIT\_B\_DATA01** 0x42A101D8U, 0x1-U, 0x42A10550U, 0x0U, 0x42A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_XSPI\_SLV\_DATA01** 0x42A101D8U, 0x2U, 0x42A10A0-CU, 0x0U, 0x42A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_QTIMER8\_TIMER1** 0x42A101D8U, 0x3U, 0x42A108-ACU, 0x1U, 0x42A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_LPSPi4\_PCS0** 0x42A101D8U, 0x4U, 0x42A10624U, 0x1U, 0x42A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_GPIO5\_IO19** 0x42A101D8U, 0x5U, 0, 0, 0x42A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_LPUART5\_RX** 0x42A101D8U, 0x6U, 0x42A1069CU, 0x4U, 0x42A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_09\_NETC\_TMR\_PP1** 0x42A101D8U, 0x9U, 0, 0, 0x42-A10420U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_NETC\_EMDIO** 0x42A101DCU, 0xAU, 0x42A10798U, 0x5U, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_ECAT\_MDIO** 0x42A101DCU, 0xCU, 0x42A104ECU, 0x1U, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_USDHC2\_DATA6** 0x42A101DCU, 0x0U, 0, 0, 0x42-A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_FLEXSPi1\_BUS2BIT\_B\_DATA02** 0x42A101DCU, 0x1-U, 0x42A10554U, 0x0U, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_XSPI\_SLV\_DATA02** 0x42A101DCU, 0x2U, 0x42A10-A10U, 0x0U, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_QTIMER8\_TIMER2** 0x42A101DCU, 0x3U, 0, 0, 0x42-A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_LPSPi4\_SDO** 0x42A101DCU, 0x4U, 0x42A10630U, 0x1-U, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_GPIO5\_IO20** 0x42A101DCU, 0x5U, 0, 0, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_LPUART5\_DCD\_B** 0x42A101DCU, 0x6U, 0x42A10690-U, 0x1U, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_NETC\_TMR\_TRIG2** 0x42A101DCU, 0x8U, 0x42A107-D0U, 0x3U, 0x42A10424U
- #define **IOMUXC\_GPIO\_SD\_B2\_10\_NETC\_TMR\_PP3** 0x42A101DCU, 0x9U, 0, 0, 0x42-A10424U

- #define **IOMUXC\_GPIO\_SD\_B2\_11\_USDHC2\_DATA7** 0x42A101E0U, 0x0U, 0, 0, 0x42-A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_FLEXSPI1\_BUS2BIT\_B\_DATA03** 0x42A101E0U, 0x1-U, 0x42A10558U, 0x0U, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_XSPI\_SLV\_DATA03** 0x42A101E0U, 0x2U, 0x42A10-A14U, 0x0U, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_QTIMER8\_TIMER3** 0x42A101E0U, 0x3U, 0, 0, 0x42-A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_LPSPi4\_SDI** 0x42A101E0U, 0x4U, 0x42A1062CU, 0x1-U, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_GPIO5\_IO21** 0x42A101E0U, 0x5U, 0, 0, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_LPUART5\_DSR\_B** 0x42A101E0U, 0x6U, 0x42A10694-U, 0x1U, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_SFA\_ATX\_CLK\_OUT** 0x42A101E0U, 0x7U, 0, 0, 0x42-A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_NETC\_TMR\_TRIG1** 0x42A101E0U, 0x8U, 0x42A107C-CU, 0x3U, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_NETC\_EMDC** 0x42A101E0U, 0xAU, 0, 0, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_11\_ECAT\_MCLK** 0x42A101E0U, 0xCU, 0, 0, 0x42A10428U
- #define **IOMUXC\_GPIO\_SD\_B2\_12\_DUMMY\_FLEXSPI1\_BUS2BIT\_A\_DQS** 0x42A101E4-U, 0x0U, 0x42A10544U, 0x0U, 0x42A1042CU
- #define **IOMUXC\_GPIO\_SD\_B2\_12\_DUMMY\_FLEXSPI1\_BUS2BIT\_B\_DQS** 0x42A101E4-U, 0x1U, 0x42A10548U, 0x1U, 0x42A1042CU
- #define **IOMUXC\_GPIO\_SD\_B2\_12\_DUMMY\_GPIO5\_IO22** 0x42A101E4U, 0x5U, 0, 0, 0x42A1042CU
- #define **IOMUXC\_GPIO\_B1\_00\_NETC\_PINMUX\_ETH1\_TXD00** 0x42A101E8U, 0x0U, 0, 0, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_00\_ADC2\_CONV\_D00** 0x42A101E8U, 0x1U, 0, 0, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_00\_SEMC\_CSX01** 0x42A101E8U, 0x2U, 0, 0, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_00\_QTIMER1\_TIMER0** 0x42A101E8U, 0x3U, 0x42A10858U, 0x2U, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_00\_XBAR1\_XBAR\_INOUT26** 0x42A101E8U, 0x4U, 0x42-A10960U, 0x1U, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_00\_GPIO6\_IO00** 0x42A101E8U, 0x5U, 0, 0, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_00 TPM5\_CH00** 0x42A101E8U, 0x6U, 0, 0, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_00\_NETC\_PINMUX\_ETH4\_TXD00** 0x42A101E8U, 0x8U, 0, 0, 0x42A10430U
- #define **IOMUXC\_GPIO\_B1\_01\_NETC\_PINMUX\_ETH1\_TXD01** 0x42A101ECU, 0x0U, 0, 0, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_01\_ADC2\_CONV\_D01** 0x42A101ECU, 0x1U, 0, 0, 0x42A10434-U
- #define **IOMUXC\_GPIO\_B1\_01\_SEMC\_CSX02** 0x42A101ECU, 0x2U, 0, 0, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_01\_QTIMER1\_TIMER1** 0x42A101ECU, 0x3U, 0x42A1085CU, 0x2U, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_01\_XBAR1\_XBAR\_INOUT27** 0x42A101ECU, 0x4U, 0x42-A10964U, 0x1U, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_01\_GPIO6\_IO01** 0x42A101ECU, 0x5U, 0, 0, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_01 TPM5\_CH01** 0x42A101ECU, 0x6U, 0, 0, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_01\_NETC\_PINMUX\_ETH4\_TXD01** 0x42A101ECU, 0x8U, 0, 0, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_01\_SAI4\_RX\_DATA00** 0x42A101ECU, 0x9U, 0x42A108B8U,

- 0x1U, 0x42A10434U
- #define **IOMUXC\_GPIO\_B1\_02\_NETC\_PINMUX\_ETH1\_TX\_EN** 0x42A101F0U, 0x0U, 0, 0,  
0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_ADC2\_CONV\_D02** 0x42A101F0U, 0x1U, 0, 0, 0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_LPI2C6\_SCL** 0x42A101F0U, 0x2U, 0x42A10600U, 0x0U,  
0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_QTIMER1\_TIMER2** 0x42A101F0U, 0x3U, 0x42A10860U,  
0x1U, 0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_XBAR1\_XBAR\_INOUT28** 0x42A101F0U, 0x4U, 0x42-  
A10968U, 0x1U, 0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_GPIO6\_IO02** 0x42A101F0U, 0x5U, 0, 0, 0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_TPM5\_CH02** 0x42A101F0U, 0x6U, 0, 0, 0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_FLEXSPI1\_BUS2BIT\_B\_SS1\_B** 0x42A101F0U, 0x7U, 0, 0,  
0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_NETC\_PINMUX\_ETH4\_TX\_EN** 0x42A101F0U, 0x8U, 0, 0,  
0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_LPUART11\_TX** 0x42A101F0U, 0x9U, 0x42A10678U, 0x1U,  
0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_02\_SAI4\_RX\_DATA01** 0x42A101F0U, 0xCU, 0x42A108BCU,  
0x0U, 0x42A10438U
- #define **IOMUXC\_GPIO\_B1\_03\_NETC\_PINMUX\_ETH1\_TX\_CLK** 0x42A101F4U, 0x0U, 0,  
0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_ADC2\_CONV\_D03** 0x42A101F4U, 0x1U, 0, 0, 0x42A1043C-  
U
- #define **IOMUXC\_GPIO\_B1\_03\_LPI2C6\_SDA** 0x42A101F4U, 0x2U, 0x42A10604U, 0x0U,  
0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_QTIMER2\_TIMER0** 0x42A101F4U, 0x3U, 0x42A10864U,  
0x2U, 0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_XBAR1\_XBAR\_INOUT29** 0x42A101F4U, 0x4U, 0x42-  
A1096CU, 0x1U, 0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_GPIO6\_IO03** 0x42A101F4U, 0x5U, 0, 0, 0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_TPM5\_CH03** 0x42A101F4U, 0x6U, 0, 0, 0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_FLEXSPI1\_BUS2BIT\_B\_DQS** 0x42A101F4U, 0x7U, 0x42-  
A10548U, 0x2U, 0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_NETC\_PINMUX\_ETH4\_TX\_CLK** 0x42A101F4U, 0x8U,  
0x42A10854U, 0x2U, 0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_LPUART11\_RX** 0x42A101F4U, 0x9U, 0x42A10674U, 0x1U,  
0x42A1043CU
- #define **IOMUXC\_GPIO\_B1\_03\_SAI4\_RX\_DATA02** 0x42A101F4U, 0xCU, 0, 0, 0x42A1043-  
CU
- #define **IOMUXC\_GPIO\_B1\_04\_NETC\_PINMUX\_ETH1\_RXD00** 0x42A101F8U, 0x0U, 0, 0,  
0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_ADC2\_CONV\_D04** 0x42A101F8U, 0x1U, 0, 0, 0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_LPUART9\_RX** 0x42A101F8U, 0x2U, 0x42A106C8U, 0x1U,  
0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_QTIMER2\_TIMER1** 0x42A101F8U, 0x3U, 0x42A10868U,  
0x2U, 0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_XBAR1\_XBAR\_INOUT30** 0x42A101F8U, 0x4U, 0x42-  
A10970U, 0x1U, 0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_GPIO6\_IO04** 0x42A101F8U, 0x5U, 0, 0, 0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_TPM5\_EXTCLK** 0x42A101F8U, 0x6U, 0, 0, 0x42A10440U

- #define **IOMUXC\_GPIO\_B1\_04\_FLEXSPI1\_BUS2BIT\_B\_SS0\_B** 0x42A101F8U, 0x7U, 0, 0, 0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_NETC\_PINMUX\_ETH4\_RXD00** 0x42A101F8U, 0x8U, 0x42A10844U, 0x2U, 0x42A10440U
- #define **IOMUXC\_GPIO\_B1\_04\_SAI4\_RX\_DATA03** 0x42A101F8U, 0xCU, 0, 0, 0x42A10440-U
- #define **IOMUXC\_GPIO\_B1\_05\_NETC\_PINMUX\_ETH1\_RXD01** 0x42A101FCU, 0x0U, 0, 0, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_ADC2\_CONV\_D05** 0x42A101FCU, 0x1U, 0, 0, 0x42A10444-U
- #define **IOMUXC\_GPIO\_B1\_05\_LPUART9\_CTS\_B** 0x42A101FCU, 0x2U, 0, 0, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_QTIMER2\_TIMER2** 0x42A101FCU, 0x3U, 0x42A1086CU, 0x1U, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_XBAR1\_XBAR\_INOUT31** 0x42A101FCU, 0x4U, 0x42A10974U, 0x1U, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_GPIO6\_IO05** 0x42A101FCU, 0x5U, 0, 0, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_TPM6\_EXTCLK** 0x42A101FCU, 0x6U, 0, 0, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_FLEXSPI1\_BUS2BIT\_B\_SCLK** 0x42A101FCU, 0x7U, 0x42A1056CU, 0x1U, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_NETC\_PINMUX\_ETH4\_RXD01** 0x42A101FCU, 0x8U, 0x42A10848U, 0x2U, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_05\_SAI4\_MCLK** 0x42A101FCU, 0xCU, 0x42A108B0U, 0x1U, 0x42A10444U
- #define **IOMUXC\_GPIO\_B1\_06\_NETC\_PINMUX\_ETH1\_RX\_DV** 0x42A10200U, 0x0U, 0, 0, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_ADC2\_CONV\_D06** 0x42A10200U, 0x1U, 0, 0, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_LPUART9\_TX** 0x42A10200U, 0x2U, 0x42A106CCU, 0x1U, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_QTIMER3\_TIMER0** 0x42A10200U, 0x3U, 0x42A10870U, 0x2U, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_XBAR1\_XBAR\_INOUT32** 0x42A10200U, 0x4U, 0x42A10978U, 0x1U, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_GPIO6\_IO06** 0x42A10200U, 0x5U, 0, 0, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_TPM6\_CH00** 0x42A10200U, 0x6U, 0, 0, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_FLEXSPI1\_BUS2BIT\_B\_DATA07** 0x42A10200U, 0x7U, 0x42A10568U, 0x1U, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_NETC\_PINMUX\_ETH4\_RX\_DV** 0x42A10200U, 0x8U, 0x42A1083CU, 0x2U, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_06\_SAI4\_RX\_BCLK** 0x42A10200U, 0xCU, 0x42A108B4U, 0x1U, 0x42A10448U
- #define **IOMUXC\_GPIO\_B1\_07\_NETC\_PINMUX\_ETH1\_RXD02** 0x42A10204U, 0x0U, 0, 0, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_ADC2\_CONV\_D07** 0x42A10204U, 0x1U, 0, 0, 0x42A1044C-U
- #define **IOMUXC\_GPIO\_B1\_07\_LPUART9\_RTS\_B** 0x42A10204U, 0x2U, 0, 0, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_QTIMER3\_TIMER1** 0x42A10204U, 0x3U, 0x42A10874U, 0x2U, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_XBAR1\_XBAR\_INOUT33** 0x42A10204U, 0x4U, 0x42A1097CU, 0x1U, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_GPIO6\_IO07** 0x42A10204U, 0x5U, 0, 0, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_TPM6\_CH01** 0x42A10204U, 0x6U, 0, 0, 0x42A1044CU

- #define **IOMUXC\_GPIO\_B1\_07\_FLEXSPI1\_BUS2BIT\_B\_DATA06** 0x42A10204U, 0x7U, 0x42A10564U, 0x1U, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_NETC\_PINMUX\_ETH4\_TXD02** 0x42A10204U, 0x8U, 0x42A10840U, 0x2U, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_LPSPI6\_SDI** 0x42A10204U, 0x9U, 0x42A10664U, 0x2U, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_07\_SAI4\_RX\_SYNC** 0x42A10204U, 0xCU, 0x42A108C0U, 0x1-U, 0x42A1044CU
- #define **IOMUXC\_GPIO\_B1\_08\_SAI4\_TX\_BCLK** 0x42A10208U, 0xCU, 0x42A108C4U, 0x1-U, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_NETC\_PINMUX\_ETH1\_TXD03** 0x42A10208U, 0x0U, 0, 0, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_ADC2\_CONV\_RDY\_CLK** 0x42A10208U, 0x1U, 0, 0, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_USDHC1\_CD\_B** 0x42A10208U, 0x2U, 0x42A10924U, 0x1U, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_QTIMER3\_TIMER2** 0x42A10208U, 0x3U, 0x42A10878U, 0x1U, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_XBAR1\_XBAR\_INOUT36** 0x42A10208U, 0x4U, 0x42A10988U, 0x1U, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_GPIO6\_IO08** 0x42A10208U, 0x5U, 0, 0, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08 TPM6\_CH02** 0x42A10208U, 0x6U, 0, 0, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_FLEXSPI1\_BUS2BIT\_B\_DATA05** 0x42A10208U, 0x7U, 0x42A10560U, 0x1U, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_NETC\_PINMUX\_ETH4\_TXD03** 0x42A10208U, 0x8U, 0, 0, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_08\_LPSPI6\_SDO** 0x42A10208U, 0x9U, 0x42A10668U, 0x2U, 0x42A10450U
- #define **IOMUXC\_GPIO\_B1\_09\_NETC\_PINMUX\_ETH1\_RXD02** 0x42A1020CU, 0x0U, 0, 0, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_USDHC1\_WP** 0x42A1020CU, 0x2U, 0x42A10928U, 0x1U, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_QTIMER4\_TIMER0** 0x42A1020CU, 0x3U, 0x42A1087CU, 0x2U, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_XBAR1\_XBAR\_INOUT37** 0x42A1020CU, 0x4U, 0x42A1098CU, 0x1U, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_GPIO6\_IO09** 0x42A1020CU, 0x5U, 0, 0, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09 TPM6\_CH03** 0x42A1020CU, 0x6U, 0, 0, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_FLEXSPI1\_BUS2BIT\_B\_DATA04** 0x42A1020CU, 0x7U, 0x42A1055CU, 0x1U, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_NETC\_PINMUX\_ETH4\_RXD02** 0x42A1020CU, 0x8U, 0x42A1084CU, 0x2U, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_LPSPI6\_PCS1** 0x42A1020CU, 0x9U, 0x42A10654U, 0x2U, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_09\_SAI4\_TX\_SYNC** 0x42A1020CU, 0xCU, 0x42A108C8U, 0x1-U, 0x42A10454U
- #define **IOMUXC\_GPIO\_B1\_10\_NETC\_PINMUX\_ETH1\_RXD03** 0x42A10210U, 0x0U, 0, 0, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_10\_USDHC1\_RESET\_B** 0x42A10210U, 0x2U, 0, 0, 0x42A10458U

- #define **IOMUXC\_GPIO\_B1\_10\_QTIMER4\_TIMER1** 0x42A10210U, 0x3U, 0x42A10880U, 0x2U, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_10\_XBAR1\_XBAR\_INOUT34** 0x42A10210U, 0x4U, 0x42A10980U, 0x1U, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_10\_GPIO6\_IO10** 0x42A10210U, 0x5U, 0, 0, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_10\_FLEXSPI1\_BUS2BIT\_B\_DATA03** 0x42A10210U, 0x7U, 0x42A10558U, 0x1U, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_10\_NETC\_PINMUX\_ETH4\_RXD03** 0x42A10210U, 0x8U, 0x42A10850U, 0x2U, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_10\_LPSPI6\_PCS2** 0x42A10210U, 0x9U, 0x42A10658U, 0x2U, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_10\_SAI4\_TX\_DATA00** 0x42A10210U, 0xCU, 0, 0, 0x42A10458U
- #define **IOMUXC\_GPIO\_B1\_11\_NETC\_PINMUX\_ETH1\_RX\_CLK** 0x42A10214U, 0x0U, 0, 0, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_11\_QTIMER4\_TIMER2** 0x42A10214U, 0x3U, 0x42A10884U, 0x1U, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_11\_XBAR1\_XBAR\_INOUT35** 0x42A10214U, 0x4U, 0x42A10984U, 0x1U, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_11\_GPIO6\_IO11** 0x42A10214U, 0x5U, 0, 0, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_11\_FLEXSPI1\_BUS2BIT\_B\_DATA02** 0x42A10214U, 0x7U, 0x42A10554U, 0x1U, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_11\_NETC\_PINMUX\_ETH4\_RX\_CLK** 0x42A10214U, 0x8U, 0x42A10838U, 0x2U, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_11\_LPSPI6\_PCS3** 0x42A10214U, 0x9U, 0x42A1065CU, 0x1U, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_11\_SAI4\_TX\_DATA01** 0x42A10214U, 0xCU, 0, 0, 0x42A1045CU
- #define **IOMUXC\_GPIO\_B1\_12\_SAI4\_TX\_DATA02** 0x42A10218U, 0xCU, 0, 0, 0x42A10460U
- #define **IOMUXC\_GPIO\_B1\_12\_NETC\_PINMUX\_ETH1\_RX\_ER** 0x42A10218U, 0x0U, 0, 0, 0x42A10460U
- #define **IOMUXC\_GPIO\_B1\_12\_NETC\_EMDIO** 0x42A10218U, 0x1U, 0x42A10798U, 0x6U, 0x42A10460U
- #define **IOMUXC\_GPIO\_B1\_12\_GPIO6\_IO12** 0x42A10218U, 0x5U, 0, 0, 0x42A10460U
- #define **IOMUXC\_GPIO\_B1\_12\_FLEXSPI1\_BUS2BIT\_B\_DATA01** 0x42A10218U, 0x7U, 0x42A10550U, 0x1U, 0x42A10460U
- #define **IOMUXC\_GPIO\_B1\_12\_NETC\_PINMUX\_ETH4\_RX\_ER** 0x42A10218U, 0x8U, 0, 0, 0x42A10460U
- #define **IOMUXC\_GPIO\_B1\_12\_LPSPI6\_PCS0** 0x42A10218U, 0x9U, 0x42A10650U, 0x2U, 0x42A10460U
- #define **IOMUXC\_GPIO\_B1\_13\_NETC\_PINMUX\_ETH1\_TX\_ER** 0x42A1021CU, 0x0U, 0, 0, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_NETC\_EMDC** 0x42A1021CU, 0x1U, 0, 0, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_USDHCI\_VSELECT** 0x42A1021CU, 0x2U, 0, 0, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_CCM\_ENET\_REF\_CLK\_25M** 0x42A1021CU, 0x3U, 0, 0, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_GPIO6\_IO13** 0x42A1021CU, 0x5U, 0, 0, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_FLEXSPI1\_BUS2BIT\_B\_DATA00** 0x42A1021CU, 0x7U,

- 0x42A1054CU, 0x1U, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_NETC\_PINMUX\_ETH4\_TX\_ER** 0x42A1021CU, 0x8U, 0, 0, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_LPSPI6\_SCK** 0x42A1021CU, 0x9U, 0x42A10660U, 0x2U, 0x42A10464U
- #define **IOMUXC\_GPIO\_B1\_13\_SAI4\_TX\_DATA03** 0x42A1021CU, 0x1U, 0, 0, 0x42A10464U
- #define **IOMUXC\_GPIO\_B2\_00\_NETC\_ETH2\_SLV\_MDIO** 0x42A10220U, 0xAU, 0x42A107A8U, 0x2U, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_ECAT\_CLK\_ECAT\_CLK25** 0x42A10220U, 0x1U, 0, 0, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_NETC\_ETH1 CRS** 0x42A10220U, 0x0U, 0, 0, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_SEMC\_CSX03** 0x42A10220U, 0x1U, 0, 0, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_LPIT3\_TRIGGER00** 0x42A10220U, 0x2U, 0, 0, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_SAI4\_MCLK** 0x42A10220U, 0x4U, 0x42A108B0U, 0x2U, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_GPIO6\_IO14** 0x42A10220U, 0x5U, 0, 0, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_NETC\_ETH4 CRS** 0x42A10220U, 0x8U, 0x42A107C0U, 0x2U, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_00\_LPSPI6\_SDI** 0x42A10220U, 0x9U, 0x42A10664U, 0x3U, 0x42A10468U
- #define **IOMUXC\_GPIO\_B2\_01\_NETC\_ETH2\_SLV\_MDC** 0x42A10224U, 0xAU, 0x42A107A4U, 0x2U, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_NETC\_PINMUX\_ETH2\_RX\_ER** 0x42A10224U, 0xBU, 0x42A10800U, 0x1U, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_ECAT\_RX\_ER\_1** 0x42A10224U, 0x1U, 0x42A104E0U, 0x2U, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_NETC\_ETH1\_COL** 0x42A10224U, 0x0U, 0, 0, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_LPIT3\_TRIGGER01** 0x42A10224U, 0x2U, 0, 0, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_SAI4\_TX\_BCLK** 0x42A10224U, 0x4U, 0x42A108C4U, 0x2U, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_GPIO6\_IO15** 0x42A10224U, 0x5U, 0, 0, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_FLEXSPI1\_BUS2BIT\_A\_SS1\_B** 0x42A10224U, 0x6U, 0, 0, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_NETC\_ETH4\_COL** 0x42A10224U, 0x8U, 0x42A107BCU, 0x2U, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_01\_LPSPI6\_SDO** 0x42A10224U, 0x9U, 0x42A10668U, 0x3U, 0x42A1046CU
- #define **IOMUXC\_GPIO\_B2\_02\_EWM\_EWM\_OUT\_B** 0x42A10228U, 0xAU, 0, 0, 0x42A10470U
- #define **IOMUXC\_GPIO\_B2\_02\_NETC\_PINMUX\_ETH2\_RXD02** 0x42A10228U, 0xBU, 0x42A1080CU, 0x2U, 0x42A10470U
- #define **IOMUXC\_GPIO\_B2\_02\_ECAT\_RX\_DATA2\_1** 0x42A10228U, 0x1U, 0x42A104C8U, 0x1U, 0x42A10470U
- #define **IOMUXC\_GPIO\_B2\_02\_LPIT3\_TRIGGER02** 0x42A10228U, 0x2U, 0, 0, 0x42A10470U
- #define **IOMUXC\_GPIO\_B2\_02\_NETC\_EMDIO** 0x42A10228U, 0x3U, 0x42A10798U, 0x7U,

- 0x42A10470U
- #define **IOMUXC\_GPIO\_B2\_02\_SAI4\_TX\_SYNC** 0x42A10228U, 0x4U, 0x42A108C8U, 0x2U, 0x42A10470U
  - #define **IOMUXC\_GPIO\_B2\_02\_GPIO6\_IO16** 0x42A10228U, 0x5U, 0, 0, 0x42A10470U
  - #define **IOMUXC\_GPIO\_B2\_02\_FLEXSPI1\_BUS2BIT\_B\_SCLK** 0x42A10228U, 0x6U, 0x42-A1056CU, 0x2U, 0x42A10470U
  - #define **IOMUXC\_GPIO\_B2\_02\_CCM\_ENET\_REF\_CLK\_25M** 0x42A10228U, 0x9U, 0, 0, 0x42A10470U
  - #define **IOMUXC\_GPIO\_B2\_03\_XSPI\_SLV\_DATA04** 0x42A1022CU, 0xAU, 0x42A10A18U, 0x2U, 0x42A10474U
  - #define **IOMUXC\_GPIO\_B2\_03\_NETC\_PINMUX\_ETH2\_RXD03** 0x42A1022CU, 0xBU, 0x42A10810U, 0x2U, 0x42A10474U
  - #define **IOMUXC\_GPIO\_B2\_03\_ECAT\_RX\_DATA3\_1** 0x42A1022CU, 0xCU, 0x42A104D0U, 0x1U, 0x42A10474U
  - #define **IOMUXC\_GPIO\_B2\_03\_LPIT3\_TRIGGER03** 0x42A1022CU, 0x2U, 0, 0, 0x42-A10474U
  - #define **IOMUXC\_GPIO\_B2\_03\_NETC\_EMDC** 0x42A1022CU, 0x3U, 0, 0, 0x42A10474U
  - #define **IOMUXC\_GPIO\_B2\_03\_SAI4\_TX\_DATA00** 0x42A1022CU, 0x4U, 0, 0, 0x42A10474-U
  - #define **IOMUXC\_GPIO\_B2\_03\_GPIO6\_IO17** 0x42A1022CU, 0x5U, 0, 0, 0x42A10474U
  - #define **IOMUXC\_GPIO\_B2\_03\_FLEXSPI1\_BUS2BIT\_A\_DATA04** 0x42A1022CU, 0x7U, 0, 0, 0x42A10474U
  - #define **IOMUXC\_GPIO\_B2\_04\_XSPI\_SLV\_DATA05** 0x42A10230U, 0xAU, 0x42A10A1CU, 0x2U, 0x42A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_NETC\_PINMUX\_ETH2\_TXD02** 0x42A10230U, 0xBU, 0, 0, 0x42A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_ECAT\_TX\_DATA2\_1** 0x42A10230U, 0xCU, 0, 0, 0x42-A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_SINC1\_MOD\_CLK0** 0x42A10230U, 0x0U, 0, 0, 0x42-A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_SINC2\_MOD\_CLK0** 0x42A10230U, 0x1U, 0, 0, 0x42-A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_SINC3\_MOD\_CLK0** 0x42A10230U, 0x2U, 0, 0, 0x42-A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_SAI4\_RX\_SYNC** 0x42A10230U, 0x4U, 0x42A108C0U, 0x2-U, 0x42A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_GPIO6\_IO18** 0x42A10230U, 0x5U, 0, 0, 0x42A10478U
  - #define **IOMUXC\_GPIO\_B2\_04\_FLEXSPI1\_BUS2BIT\_A\_DATA05** 0x42A10230U, 0x7U, 0, 0, 0x42A10478U
  - #define **IOMUXC\_GPIO\_B2\_04 TPM3\_EXTCLK** 0x42A10230U, 0x8U, 0, 0, 0x42A10478U
  - #define **IOMUXC\_GPIO\_B2\_05\_XSPI\_SLV\_DATA06** 0x42A10234U, 0xAU, 0x42A10A20U, 0x2U, 0x42A1047CU
  - #define **IOMUXC\_GPIO\_B2\_05\_NETC\_PINMUX\_ETH2\_TXD03** 0x42A10234U, 0xBU, 0, 0, 0x42A1047CU
  - #define **IOMUXC\_GPIO\_B2\_05\_ECAT\_TX\_DATA3\_1** 0x42A10234U, 0xCU, 0, 0, 0x42-A1047CU
  - #define **IOMUXC\_GPIO\_B2\_05\_SINC1\_MOD\_CLK1** 0x42A10234U, 0x0U, 0, 0, 0x42A1047-CU
  - #define **IOMUXC\_GPIO\_B2\_05\_SINC2\_MOD\_CLK1** 0x42A10234U, 0x1U, 0, 0, 0x42A1047-CU

- #define **IOMUXC\_GPIO\_B2\_05\_SINC3\_MOD\_CLK1** 0x42A10234U, 0x2U, 0, 0, 0x42A1047-CU
- #define **IOMUXC\_GPIO\_B2\_05\_SAI4\_RX\_BCLK** 0x42A10234U, 0x4U, 0x42A108B4U, 0x2-U, 0x42A1047CU
- #define **IOMUXC\_GPIO\_B2\_05\_GPIO6\_IO19** 0x42A10234U, 0x5U, 0, 0, 0x42A1047CU
- #define **IOMUXC\_GPIO\_B2\_05\_MIC\_CLK** 0x42A10234U, 0x6U, 0, 0, 0x42A1047CU
- #define **IOMUXC\_GPIO\_B2\_05\_FLEXSPI1\_BUS2BIT\_A\_DATA06** 0x42A10234U, 0x7U, 0, 0, 0x42A1047CU
- #define **IOMUXC\_GPIO\_B2\_05 TPM3\_CH00** 0x42A10234U, 0x8U, 0, 0, 0x42A1047CU
- #define **IOMUXC\_GPIO\_B2\_06\_XSPI\_SLV\_DATA07** 0x42A10238U, 0xAU, 0x42A10A24U, 0x2U, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_NETC\_PINMUX\_ETH2\_TXD00** 0x42A10238U, 0xBU, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_ECAT\_TX\_DATA0\_1** 0x42A10238U, 0xCU, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_SINC1\_MOD\_CLK2** 0x42A10238U, 0x0U, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_SINC2\_MOD\_CLK2** 0x42A10238U, 0x1U, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_SINC3\_MOD\_CLK2** 0x42A10238U, 0x2U, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_LPUART6\_DSR\_B** 0x42A10238U, 0x3U, 0x42A106ACU, 0x1U, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_SAI4\_RX\_DATA00** 0x42A10238U, 0x4U, 0x42A108B8U, 0x2U, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_GPIO6\_IO20** 0x42A10238U, 0x5U, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_FLEXSPI1\_BUS2BIT\_A\_DATA07** 0x42A10238U, 0x7U, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06 TPM3\_CH01** 0x42A10238U, 0x8U, 0, 0, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_06\_LPUART11\_TX** 0x42A10238U, 0x9U, 0x42A10678U, 0x2U, 0x42A10480U
- #define **IOMUXC\_GPIO\_B2\_07\_XSPI\_SLV\_DQS** 0x42A1023CU, 0xAU, 0x42A10A04U, 0x1-U, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_NETC\_PINMUX\_ETH2\_TXD01** 0x42A1023CU, 0xBU, 0, 0, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_ECAT\_TX\_DATA1\_1** 0x42A1023CU, 0xCU, 0, 0, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_QTIMER5\_TIMER0** 0x42A1023CU, 0x0U, 0x42A10888U, 0x2U, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_LPUART6\_DCD\_B** 0x42A1023CU, 0x3U, 0x42A106A8U, 0x1U, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_SAI4\_TX\_DATA01** 0x42A1023CU, 0x4U, 0, 0, 0x42A10484-U
- #define **IOMUXC\_GPIO\_B2\_07\_GPIO6\_IO21** 0x42A1023CU, 0x5U, 0, 0, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_SAI4\_RX\_DATA01** 0x42A1023CU, 0x6U, 0x42A108BCU, 0x1U, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_FLEXSPI1\_BUS2BIT\_A\_DQS** 0x42A1023CU, 0x7U, 0x42A10544U, 0x1U, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07 TPM3\_CH02** 0x42A1023CU, 0x8U, 0, 0, 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_07\_LPUART11\_RX** 0x42A1023CU, 0x9U, 0x42A10674U, 0x2U,

- 0x42A10484U
- #define **IOMUXC\_GPIO\_B2\_08\_XSPI\_SLV\_CLK** 0x42A10240U, 0xAU, 0x42A10A28U, 0x2-U, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_NETC\_PINMUX\_ETH2\_TX\_EN** 0x42A10240U, 0xBU, 0, 0, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_ECAT\_TX\_EN\_1** 0x42A10240U, 0xCU, 0, 0, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_QTIMER5\_TIMER1** 0x42A10240U, 0x0U, 0x42A1088CU, 0x2U, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_SINC2\_EMCLK02** 0x42A10240U, 0x1U, 0x42A10900U, 0x2U, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_LPUART6\_RI\_B** 0x42A10240U, 0x4U, 0x42A106B0U, 0x1-U, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_GPIO6\_IO22** 0x42A10240U, 0x5U, 0, 0, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_LPI2C6\_SCL** 0x42A10240U, 0x6U, 0x42A10600U, 0x1U, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_FLEXSPI1\_BUS2BIT\_A\_SCLK** 0x42A10240U, 0x7U, 0, 0, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08 TPM3\_CH03** 0x42A10240U, 0x8U, 0, 0, 0x42A10488U
  - #define **IOMUXC\_GPIO\_B2\_08\_SPDIF\_IN** 0x42A10240U, 0x9U, 0x42A10908U, 0x2U, 0x42-A10488U
  - #define **IOMUXC\_GPIO\_B2\_09\_XSPI\_SLV\_CS** 0x42A10244U, 0xAU, 0x42A10A00U, 0x2U, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_NETC\_PINMUX\_ETH2\_TX\_CLK** 0x42A10244U, 0xBU, 0x42A10814U, 0x2U, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_ECAT\_TX\_CLK\_1** 0x42A10244U, 0xCU, 0x42A104E8U, 0x2U, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_QTIMER5\_TIMER2** 0x42A10244U, 0xOU, 0x42A10890U, 0x1U, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_SINC2\_EMBIT02** 0x42A10244U, 0x1U, 0x42A108F4U, 0x2-U, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_LPUART6\_DTR\_B** 0x42A10244U, 0x4U, 0, 0, 0x42A1048C-U
  - #define **IOMUXC\_GPIO\_B2\_09\_GPIO6\_IO23** 0x42A10244U, 0x5U, 0, 0, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_LPI2C6\_SDA** 0x42A10244U, 0x6U, 0x42A10604U, 0x1U, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_FLEXSPI1\_BUS2BIT\_A\_SS0\_B** 0x42A10244U, 0x7U, 0, 0, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09 TPM4\_EXTCLK** 0x42A10244U, 0x8U, 0, 0, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_09\_SPDIF\_OUT** 0x42A10244U, 0x9U, 0, 0, 0x42A1048CU
  - #define **IOMUXC\_GPIO\_B2\_10\_XSPI\_SLV\_DATA00** 0x42A10248U, 0xAU, 0x42A10A08U, 0x1U, 0x42A10490U
  - #define **IOMUXC\_GPIO\_B2\_10\_NETC\_PINMUX\_ETH2\_RXD00** 0x42A10248U, 0xBU, 0x42A10804U, 0x2U, 0x42A10490U
  - #define **IOMUXC\_GPIO\_B2\_10\_ECAT\_RX\_DATA0\_1** 0x42A10248U, 0xCU, 0x42A104B8U, 0x2U, 0x42A10490U
  - #define **IOMUXC\_GPIO\_B2\_10\_MIC\_BITSTREAM00** 0x42A10248U, 0xOU, 0x42A106D0U, 0x3U, 0x42A10490U
  - #define **IOMUXC\_GPIO\_B2\_10\_SINC2\_EMCLK03** 0x42A10248U, 0x1U, 0x42A10904U, 0x1U, 0x42A10490U
  - #define **IOMUXC\_GPIO\_B2\_10\_CAN3\_TX** 0x42A10248U, 0x2U, 0, 0, 0x42A10490U
  - #define **IOMUXC\_GPIO\_B2\_10\_LPUART8\_CTS\_B** 0x42A10248U, 0x3U, 0x42A106BCU,

- 0x1U, 0x42A10490U
- #define **IOMUXC\_GPIO\_B2\_10\_LPUART6\_TX** 0x42A10248U, 0x4U, 0x42A106B8U, 0x2U, 0x42A10490U
- #define **IOMUXC\_GPIO\_B2\_10\_GPIO6\_IO24** 0x42A10248U, 0x5U, 0, 0, 0x42A10490U
- #define **IOMUXC\_GPIO\_B2\_10\_LPI2C4\_SCL** 0x42A10248U, 0x6U, 0x42A105F0U, 0x1U, 0x42A10490U
- #define **IOMUXC\_GPIO\_B2\_10\_FLEXSPI1\_BUS2BIT\_A\_DATA00** 0x42A10248U, 0x7U, 0, 0, 0x42A10490U
- #define **IOMUXC\_GPIO\_B2\_10 TPM4\_CH00** 0x42A10248U, 0x8U, 0, 0, 0x42A10490U
- #define **IOMUXC\_GPIO\_B2\_10\_LPSPI4\_SCK** 0x42A10248U, 0x9U, 0x42A10628U, 0x2U, 0x42A10490U
- #define **IOMUXC\_GPIO\_B2\_11\_XSPI\_SLV\_DATA01** 0x42A1024CU, 0xAU, 0x42A10A0CU, 0x1U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_NETC\_PINMUX\_ETH2\_RXD01** 0x42A1024CU, 0xBU, 0x42A10808U, 0x2U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_ECAT\_RX\_DATA1\_1** 0x42A1024CU, 0xCU, 0x42A104C0U, 0x2U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_MIC\_BITSTREAM01** 0x42A1024CU, 0x0U, 0x42A106D4U, 0x3U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_SINC2\_EMBIT03** 0x42A1024CU, 0x1U, 0x42A108F8U, 0x1-U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_CAN3\_RX** 0x42A1024CU, 0x2U, 0x42A104A8U, 0x1U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_LPUART8\_RTS\_B** 0x42A1024CU, 0x3U, 0, 0, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_LPUART6\_RX** 0x42A1024CU, 0x4U, 0x42A106B4U, 0x2U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_GPIO6\_IO25** 0x42A1024CU, 0x5U, 0, 0, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_LPI2C4\_SDA** 0x42A1024CU, 0x6U, 0x42A105F4U, 0x1U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_FLEXSPI1\_BUS2BIT\_A\_DATA01** 0x42A1024CU, 0x7U, 0, 0, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11 TPM4\_CH01** 0x42A1024CU, 0x8U, 0, 0, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_11\_LPSPI4\_SDI** 0x42A1024CU, 0x9U, 0x42A1062CU, 0x2U, 0x42A10494U
- #define **IOMUXC\_GPIO\_B2\_12\_MIC\_BITSTREAM02** 0x42A10250U, 0x0U, 0x42A106D8U, 0x3U, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_SINC\_FILTER\_GLUE2\_BREAK** 0x42A10250U, 0x1U, 0, 0, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_LPUART8\_TX** 0x42A10250U, 0x2U, 0x42A106C4U, 0x2U, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_LPUART6\_CTS\_B** 0x42A10250U, 0x4U, 0x42A106A4U, 0x2U, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_GPIO6\_IO26** 0x42A10250U, 0x5U, 0, 0, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_CAN3\_TX** 0x42A10250U, 0x6U, 0, 0, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_FLEXSPI1\_BUS2BIT\_A\_DATA02** 0x42A10250U, 0x7U, 0, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12 TPM4\_CH02** 0x42A10250U, 0x8U, 0, 0, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_LPSPI4\_SDO** 0x42A10250U, 0x9U, 0x42A10630U, 0x2U, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_XSPI\_SLV\_DATA02** 0x42A10250U, 0xAU, 0x42A10A10U, 0x1U, 0x42A10498U

- #define **IOMUXC\_GPIO\_B2\_12\_NETC\_PINMUX\_ETH2\_RX\_DV** 0x42A10250U, 0xBU, 0x42A107FCU, 0x2U, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_12\_ECAT\_RX\_DV\_1** 0x42A10250U, 0xCU, 0x42A104D8U, 0x2-U, 0x42A10498U
- #define **IOMUXC\_GPIO\_B2\_13\_MIC\_BITSTREAM03** 0x42A10254U, 0x0U, 0x42A106DCU, 0x3U, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_SINC2\_EMCLK00** 0x42A10254U, 0x1U, 0x42A108FCU, 0x1U, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_LPUART8\_RX** 0x42A10254U, 0x2U, 0x42A106C0U, 0x2U, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_LPUART6\_RTS\_B** 0x42A10254U, 0x4U, 0, 0, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_GPIO6\_IO27** 0x42A10254U, 0x5U, 0, 0, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_CAN3\_RX** 0x42A10254U, 0x6U, 0x42A104A8U, 0x2U, 0x42-A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_FLEXSPI1\_BUS2BIT\_A\_DATA03** 0x42A10254U, 0x7U, 0, 0, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_TPM4\_CH03** 0x42A10254U, 0x8U, 0, 0, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_LPSPi4\_PCS0** 0x42A10254U, 0x9U, 0x42A10624U, 0x2U, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_XSPI\_SLV\_DATA03** 0x42A10254U, 0xAU, 0x42A10A14U, 0x1U, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_NETC\_PINMUX\_ETH2\_RX\_CLK** 0x42A10254U, 0xBU, 0x42A107F8U, 0x3U, 0x42A1049CU
- #define **IOMUXC\_GPIO\_B2\_13\_ECAT\_RX\_CLK\_1** 0x42A10254U, 0xCU, 0x42A104B0U, 0x1U, 0x42A1049CU
- #define **IOMUXC\_GPIO\_AON\_00\_SRC\_BOOT\_MODE00** 0x443C0000U, 0x0U, 0, 0, 0x443-C0074U
- #define **IOMUXC\_GPIO\_AON\_00\_CAN1\_TX** 0x443C0000U, 0x1U, 0, 0, 0x443C0074U
- #define **IOMUXC\_GPIO\_AON\_00\_LPTMR1\_ALT1** 0x443C0000U, 0x4U, 0x443C012CU, 0x0U, 0x443C0074U
- #define **IOMUXC\_GPIO\_AON\_00\_GPIO1\_IO00** 0x443C0000U, 0x5U, 0, 0, 0x443C0074U
- #define **IOMUXC\_GPIO\_AON\_00\_LPUART2\_TX** 0x443C0000U, 0x6U, 0x443C0158U, 0x0U, 0x443C0074U
- #define **IOMUXC\_GPIO\_AON\_00\_TPM1\_EXTCLK** 0x443C0000U, 0x8U, 0, 0, 0x443C0074U
- #define **IOMUXC\_GPIO\_AON\_01\_SRC\_BOOT\_MODE01** 0x443C0004U, 0x0U, 0, 0, 0x443-C0078U
- #define **IOMUXC\_GPIO\_AON\_01\_CAN1\_RX** 0x443C0004U, 0x1U, 0x42A104A0U, 0x1U, 0x443C0078U
- #define **IOMUXC\_GPIO\_AON\_01\_LPTMR1\_ALT2** 0x443C0004U, 0x4U, 0x443C0130U, 0x0-U, 0x443C0078U
- #define **IOMUXC\_GPIO\_AON\_01\_GPIO1\_IO01** 0x443C0004U, 0x5U, 0, 0, 0x443C0078U
- #define **IOMUXC\_GPIO\_AON\_01\_LPUART2\_RX** 0x443C0004U, 0x6U, 0x443C0154U, 0x0-U, 0x443C0078U
- #define **IOMUXC\_GPIO\_AON\_01\_TPM1\_CH00** 0x443C0004U, 0x8U, 0, 0, 0x443C0078U
- #define **IOMUXC\_GPIO\_AON\_02\_SRC\_BOOT\_MODE02** 0x443C0008U, 0x0U, 0, 0, 0x443-C007CU
- #define **IOMUXC\_GPIO\_AON\_02\_CAN3\_TX** 0x443C0008U, 0x1U, 0, 0, 0x443C007CU
- #define **IOMUXC\_GPIO\_AON\_02\_LPSPi2\_PCS3** 0x443C0008U, 0x2U, 0x443C011CU, 0x0U, 0x443C007CU
- #define **IOMUXC\_GPIO\_AON\_02\_LPSPi2\_SDO** 0x443C0008U, 0x3U, 0x443C0128U, 0x0U, 0x443C007CU

- #define **IOMUXC\_GPIO\_AON\_02\_LPTMR1\_ALT3** 0x443C0008U, 0x4U, 0x443C0134U, 0x0U, 0x443C007CU
- #define **IOMUXC\_GPIO\_AON\_02\_GPIO1\_IO02** 0x443C0008U, 0x5U, 0, 0, 0x443C007CU
- #define **IOMUXC\_GPIO\_AON\_02\_LPUART2\_RTS\_B** 0x443C0008U, 0x6U, 0, 0, 0x443C007CU
- #define **IOMUXC\_GPIO\_AON\_02 TPM1\_CH01** 0x443C0008U, 0x8U, 0, 0, 0x443C007CU
- #define **IOMUXC\_GPIO\_AON\_02 ECAT\_CLK\_ECAT\_CLK25** 0x443C0008U, 0xCU, 0, 0, 0x443C007CU
- #define **IOMUXC\_GPIO\_AON\_03\_CAN3\_RX** 0x443C000CU, 0x1U, 0x42A104A8U, 0x3U, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_03\_LPSPI1\_PCS1** 0x443C000CU, 0x2U, 0x443C0104U, 0x0U, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_03\_LPSPI2\_SDI** 0x443C000CU, 0x3U, 0x443C0124U, 0x0U, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_03\_LPSPI1\_PCS3** 0x443C000CU, 0x4U, 0, 0, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_03\_GPIO1\_IO03** 0x443C000CU, 0x5U, 0, 0, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_03\_LPUART2\_CTS\_B** 0x443C000CU, 0x6U, 0x443C0150U, 0x0U, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_03 TPM1\_CH02** 0x443C000CU, 0x8U, 0, 0, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_03 ECAT\_LED\_STATE\_RUN** 0x443C000CU, 0xCU, 0, 0, 0x443C0080U
- #define **IOMUXC\_GPIO\_AON\_04 ECAT\_LED\_RUN** 0x443C0010U, 0xCU, 0, 0, 0x443C0084U
- #define **IOMUXC\_GPIO\_AON\_04\_LPSPI1\_SCK** 0x443C0010U, 0x0U, 0x443C0108U, 0x0U, 0x443C0084U
- #define **IOMUXC\_GPIO\_AON\_04\_SAI1\_TX\_DATA00** 0x443C0010U, 0x2U, 0, 0, 0x443C0084U
- #define **IOMUXC\_GPIO\_AON\_04\_SAI1\_RX\_DATA01** 0x443C0010U, 0x3U, 0x443C0174U, 0x0U, 0x443C0084U
- #define **IOMUXC\_GPIO\_AON\_04\_GPIO1\_IO04** 0x443C0010U, 0x5U, 0, 0, 0x443C0084U
- #define **IOMUXC\_GPIO\_AON\_04\_LPUART7\_CTS\_B** 0x443C0010U, 0x6U, 0x443C015CU, 0x0U, 0x443C0084U
- #define **IOMUXC\_GPIO\_AON\_04 TPM1\_CH03** 0x443C0010U, 0x8U, 0, 0, 0x443C0084U
- #define **IOMUXC\_GPIO\_AON\_05\_LPSPI1\_PCS0** 0x443C0014U, 0x0U, 0x443C0100U, 0x0U, 0x443C0088U
- #define **IOMUXC\_GPIO\_AON\_05\_SAI1\_TX\_SYNC** 0x443C0014U, 0x2U, 0x443C0180U, 0x0U, 0x443C0088U
- #define **IOMUXC\_GPIO\_AON\_05\_GPIO1\_IO05** 0x443C0014U, 0x5U, 0, 0, 0x443C0088U
- #define **IOMUXC\_GPIO\_AON\_05\_LPUART7\_RTS\_B** 0x443C0014U, 0x6U, 0, 0, 0x443C0088U
- #define **IOMUXC\_GPIO\_AON\_05\_NMI\_GLUE\_NMI** 0x443C0014U, 0x7U, 0, 0, 0x443C0088U
- #define **IOMUXC\_GPIO\_AON\_05 ECAT\_LED\_ERR** 0x443C0014U, 0xCU, 0, 0, 0x443C0088U
- #define **IOMUXC\_GPIO\_AON\_06\_LPSPI1\_SDO** 0x443C0018U, 0x0U, 0x443C0110U, 0x0U, 0x443C008CU
- #define **IOMUXC\_GPIO\_AON\_06\_I3C1\_PUR** 0x443C0018U, 0x1U, 0, 0, 0x443C008CU
- #define **IOMUXC\_GPIO\_AON\_06\_SAI1\_TX\_BCLK** 0x443C0018U, 0x2U, 0x443C017CU, 0x0U, 0x443C008CU
- #define **IOMUXC\_GPIO\_AON\_06\_LPI2C1\_SDA** 0x443C0018U, 0x3U, 0x443C00F4U, 0x0U, 0x443C008CU

- #define **IOMUXC\_GPIO\_AON\_06\_GPIO1\_IO06** 0x443C0018U, 0x5U, 0, 0, 0x443C008CU
- #define **IOMUXC\_GPIO\_AON\_06\_CAN1\_TX** 0x443C0018U, 0x6U, 0, 0, 0x443C008CU
- #define **IOMUXC\_GPIO\_AON\_06\_ECAT\_SDA** 0x443C0018U, 0xCU, 0x42A104F0U, 0x1U, 0x443C008CU
- #define **IOMUXC\_GPIO\_AON\_07\_ECAT\_SCL** 0x443C001CU, 0xCU, 0, 0, 0x443C0090U
- #define **IOMUXC\_GPIO\_AON\_07\_LPSPI1\_SDI** 0x443C001CU, 0x0U, 0x443C010CU, 0x0U, 0x443C0090U
- #define **IOMUXC\_GPIO\_AON\_07\_SAI1\_MCLK** 0x443C001CU, 0x2U, 0x443C0168U, 0x0U, 0x443C0090U
- #define **IOMUXC\_GPIO\_AON\_07\_LPI2C1\_SCL** 0x443C001CU, 0x3U, 0x443C00F0U, 0x0U, 0x443C0090U
- #define **IOMUXC\_GPIO\_AON\_07\_GPIO1\_IO07** 0x443C001CU, 0x5U, 0, 0, 0x443C0090U
- #define **IOMUXC\_GPIO\_AON\_07\_CAN1\_RX** 0x443C001CU, 0x6U, 0x42A104A0U, 0x2U, 0x443C0090U
- #define **IOMUXC\_GPIO\_AON\_08\_LPUART1\_TX** 0x443C0020U, 0x0U, 0, 0, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_08\_S400\_TX** 0x443C0020U, 0x1U, 0, 0, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_08\_SAI1\_RX\_DATA00** 0x443C0020U, 0x2U, 0x443C0170U, 0x0U, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_08\_SAI1\_TX\_DATA01** 0x443C0020U, 0x3U, 0, 0, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_08\_GPIO1\_IO08** 0x443C0020U, 0x5U, 0, 0, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_08\_LPI2C1\_SDA** 0x443C0020U, 0x6U, 0x443C00F4U, 0x1U, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_08\_LPSPI1\_PCS1** 0x443C0020U, 0x8U, 0x443C0104U, 0x1U, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_08\_ECAT\_LINK\_ACT00** 0x443C0020U, 0xCU, 0, 0, 0x443C0094U
- #define **IOMUXC\_GPIO\_AON\_09\_LPUART1\_RX** 0x443C0024U, 0x0U, 0, 0, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_09\_S400\_RX** 0x443C0024U, 0x1U, 0, 0, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_09\_SAI1\_RX\_BCLK** 0x443C0024U, 0x2U, 0x443C016CU, 0x0U, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_09\_LPIT1\_TRIGGER00** 0x443C0024U, 0x3U, 0, 0, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_09\_GPIO1\_IO09** 0x443C0024U, 0x5U, 0, 0, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_09\_LPI2C1\_SCL** 0x443C0024U, 0x6U, 0x443C00F0U, 0x1U, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_09\_LPSPI1\_PCS2** 0x443C0024U, 0x8U, 0, 0, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_09\_ECAT\_LINK\_ACT01** 0x443C0024U, 0xCU, 0, 0, 0x443C0098U
- #define **IOMUXC\_GPIO\_AON\_10\_JTAG\_MUX\_TRSTB** 0x443C0028U, 0x0U, 0, 0, 0x443C009CU
- #define **IOMUXC\_GPIO\_AON\_10\_LPSPI2\_PCS0** 0x443C0028U, 0x1U, 0x443C0114U, 0x0U, 0x443C009CU
- #define **IOMUXC\_GPIO\_AON\_10\_SAI1\_RX\_SYNC** 0x443C0028U, 0x2U, 0x443C0178U, 0x0U, 0x443C009CU
- #define **IOMUXC\_GPIO\_AON\_10\_LPIT1\_TRIGGER01** 0x443C0028U, 0x3U, 0, 0, 0x443C009CU
- #define **IOMUXC\_GPIO\_AON\_10 TPM2\_EXTCLK** 0x443C0028U, 0x4U, 0, 0, 0x443C009CU
- #define **IOMUXC\_GPIO\_AON\_10\_GPIO1\_IO10** 0x443C0028U, 0x5U, 0, 0, 0x443C009CU
- #define **IOMUXC\_GPIO\_AON\_10\_LPI2C1\_SCLS** 0x443C0028U, 0x6U, 0, 0, 0x443C009CU

- #define **IOMUXC\_GPIO\_AON\_11\_JTAG\_MUX\_TDO** 0x443C002CU, 0x0U, 0, 0, 0x443C00-A0U
- #define **IOMUXC\_GPIO\_AON\_11\_LPUART1\_CTS\_B** 0x443C002CU, 0x2U, 0x443C0138U, 0x0U, 0x443C00A0U
- #define **IOMUXC\_GPIO\_AON\_11\_LPIT1\_TRIGGER02** 0x443C002CU, 0x3U, 0, 0, 0x443-C00A0U
- #define **IOMUXC\_GPIO\_AON\_11 TPM2\_CH00** 0x443C002CU, 0x4U, 0, 0, 0x443C00A0U
- #define **IOMUXC\_GPIO\_AON\_11 GPIO1\_IO11** 0x443C002CU, 0x5U, 0, 0, 0x443C00A0U
- #define **IOMUXC\_GPIO\_AON\_11 LPI2C1\_SDAS** 0x443C002CU, 0x6U, 0, 0, 0x443C00A0U
- #define **IOMUXC\_GPIO\_AON\_12\_JTAG\_MUX\_TDI** 0x443C0030U, 0x0U, 0, 0, 0x443C00-A4U
- #define **IOMUXC\_GPIO\_AON\_12\_LPUART1\_RTS\_B** 0x443C0030U, 0x2U, 0, 0, 0x443C00-A4U
- #define **IOMUXC\_GPIO\_AON\_12\_LPIT1\_TRIGGER03** 0x443C0030U, 0x3U, 0, 0, 0x443-C00A4U
- #define **IOMUXC\_GPIO\_AON\_12 TPM2\_CH01** 0x443C0030U, 0x4U, 0, 0, 0x443C00A4U
- #define **IOMUXC\_GPIO\_AON\_12 GPIO1\_IO12** 0x443C0030U, 0x5U, 0, 0, 0x443C00A4U
- #define **IOMUXC\_GPIO\_AON\_12 LPI2C1\_HREQ** 0x443C0030U, 0x6U, 0, 0, 0x443C00A4U
- #define **IOMUXC\_GPIO\_AON\_12 LPSP1\_SCK** 0x443C0030U, 0x8U, 0x443C0108U, 0x1U, 0x443C00A4U
- #define **IOMUXC\_GPIO\_AON\_13\_JTAG\_MUX\_TCK** 0x443C0034U, 0x0U, 0, 0, 0x443C00-A8U
- #define **IOMUXC\_GPIO\_AON\_13\_LPUART12\_CTS\_B** 0x443C0034U, 0x2U, 0x443C0144U, 0x0U, 0x443C00A8U
- #define **IOMUXC\_GPIO\_AON\_13\_LPUART1\_DSR\_B** 0x443C0034U, 0x3U, 0x443C0140U, 0x0U, 0x443C00A8U
- #define **IOMUXC\_GPIO\_AON\_13 TPM2\_CH02** 0x443C0034U, 0x4U, 0, 0, 0x443C00A8U
- #define **IOMUXC\_GPIO\_AON\_13 GPIO1\_IO13** 0x443C0034U, 0x5U, 0, 0, 0x443C00A8U
- #define **IOMUXC\_GPIO\_AON\_13 LPTMR1\_ALT1** 0x443C0034U, 0x6U, 0x443C012CU, 0x1U, 0x443C00A8U
- #define **IOMUXC\_GPIO\_AON\_13 LPSP1\_PCS0** 0x443C0034U, 0x8U, 0x443C0100U, 0x1U, 0x443C00A8U
- #define **IOMUXC\_GPIO\_AON\_14\_JTAG\_MUX\_TMS** 0x443C0038U, 0x0U, 0, 0, 0x443C00-ACU
- #define **IOMUXC\_GPIO\_AON\_14\_LPUART12\_RTS\_B** 0x443C0038U, 0x2U, 0, 0, 0x443C00-ACU
- #define **IOMUXC\_GPIO\_AON\_14\_LPUART1\_DCD\_B** 0x443C0038U, 0x3U, 0x443C013CU, 0x0U, 0x443C00ACU
- #define **IOMUXC\_GPIO\_AON\_14 TPM2\_CH03** 0x443C0038U, 0x4U, 0, 0, 0x443C00ACU
- #define **IOMUXC\_GPIO\_AON\_14 GPIO1\_IO14** 0x443C0038U, 0x5U, 0, 0, 0x443C00ACU
- #define **IOMUXC\_GPIO\_AON\_14 LPTMR1\_ALT2** 0x443C0038U, 0x6U, 0x443C0130U, 0x1-U, 0x443C00ACU
- #define **IOMUXC\_GPIO\_AON\_14 LPSP1\_SDO** 0x443C0038U, 0x8U, 0x443C0110U, 0x1U, 0x443C00ACU
- #define **IOMUXC\_GPIO\_AON\_15\_FLEXSPI2\_BUS2BIT\_B\_DATA03** 0x443C003CU, 0x0U, 0x42A10594U, 0x2U, 0x443C00B0U
- #define **IOMUXC\_GPIO\_AON\_15\_LPSP1\_PCS1** 0x443C003CU, 0x1U, 0x443C0118U, 0x0U, 0x443C00B0U
- #define **IOMUXC\_GPIO\_AON\_15\_LPUART12\_TX** 0x443C003CU, 0x2U, 0x443C014CU, 0x0U, 0x443C00B0U
- #define **IOMUXC\_GPIO\_AON\_15\_LPUART1\_RI\_B** 0x443C003CU, 0x3U, 0, 0, 0x443C00B0-

- U
- #define **IOMUXC\_GPIO\_AON\_15\_LPI2C2\_SDA** 0x443C003CU, 0x4U, 0x443C00FCU, 0x0U, 0x443C00B0U
  - #define **IOMUXC\_GPIO\_AON\_15\_GPIO1\_IO15** 0x443C003CU, 0x5U, 0, 0, 0, 0x443C00B0U
  - #define **IOMUXC\_GPIO\_AON\_15\_LPTMR1\_ALT3** 0x443C003CU, 0x6U, 0x443C0134U, 0x1U, 0x443C00B0U
  - #define **IOMUXC\_GPIO\_AON\_15\_LPSP11\_SDI** 0x443C003CU, 0x8U, 0x443C010CU, 0x1U, 0x443C00B0U
  - #define **IOMUXC\_GPIO\_AON\_15\_I3C1\_SDA** 0x443C003CU, 0x9U, 0x443C00ECU, 0x0U, 0x443C00B0U
  - #define **IOMUXC\_GPIO\_AON\_16\_FLEXSPI2\_BUS2BIT\_B\_DATA02** 0x443C0040U, 0x0U, 0x42A10590U, 0x2U, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_LPSP12\_PCS0** 0x443C0040U, 0x1U, 0x443C0114U, 0x1U, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_LPUART12\_RX** 0x443C0040U, 0x2U, 0x443C0148U, 0x0U, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_LPUART1\_DTR\_B** 0x443C0040U, 0x3U, 0, 0, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_LPI2C2\_SCL** 0x443C0040U, 0x4U, 0x443C00F8U, 0x0U, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_GPIO1\_IO16** 0x443C0040U, 0x5U, 0, 0, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_CAN1\_TX** 0x443C0040U, 0x6U, 0, 0, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_LPUART7\_CTS\_B** 0x443C0040U, 0x8U, 0x443C015CU, 0x1U, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_16\_I3C1\_SCL** 0x443C0040U, 0x9U, 0x443C00E8U, 0x0U, 0x443C00B4U
  - #define **IOMUXC\_GPIO\_AON\_17\_FLEXSPI2\_BUS2BIT\_B\_DATA01** 0x443C0044U, 0x0U, 0x42A1058CU, 0x2U, 0x443C00B8U
  - #define **IOMUXC\_GPIO\_AON\_17\_LPSP12\_SDI** 0x443C0044U, 0x1U, 0x443C0124U, 0x1U, 0x443C00B8U
  - #define **IOMUXC\_GPIO\_AON\_17\_LPUART7\_TX** 0x443C0044U, 0x2U, 0x443C0164U, 0x0U, 0x443C00B8U
  - #define **IOMUXC\_GPIO\_AON\_17\_LPI2C2\_SDA** 0x443C0044U, 0x3U, 0x443C00FCU, 0x1U, 0x443C00B8U
  - #define **IOMUXC\_GPIO\_AON\_17\_LPUART1\_DSR\_B** 0x443C0044U, 0x4U, 0x443C0140U, 0x1U, 0x443C00B8U
  - #define **IOMUXC\_GPIO\_AON\_17\_GPIO1\_IO17** 0x443C0044U, 0x5U, 0, 0, 0x443C00B8U
  - #define **IOMUXC\_GPIO\_AON\_17\_CAN1\_RX** 0x443C0044U, 0x6U, 0x42A104A0U, 0x3U, 0x443C00B8U
  - #define **IOMUXC\_GPIO\_AON\_18\_FLEXSPI2\_BUS2BIT\_B\_DATA00** 0x443C0048U, 0x0U, 0x42A10588U, 0x2U, 0x443C00BCU
  - #define **IOMUXC\_GPIO\_AON\_18\_LPSP12\_SDO** 0x443C0048U, 0x1U, 0x443C0128U, 0x1U, 0x443C00BCU
  - #define **IOMUXC\_GPIO\_AON\_18\_LPUART7\_RX** 0x443C0048U, 0x2U, 0x443C0160U, 0x0U, 0x443C00BCU
  - #define **IOMUXC\_GPIO\_AON\_18\_LPI2C2\_SCL** 0x443C0048U, 0x3U, 0x443C00F8U, 0x1U, 0x443C00BCU
  - #define **IOMUXC\_GPIO\_AON\_18\_LPUART1\_DCD\_B** 0x443C0048U, 0x4U, 0x443C013CU, 0x1U, 0x443C00BCU
  - #define **IOMUXC\_GPIO\_AON\_18\_GPIO1\_IO18** 0x443C0048U, 0x5U, 0, 0, 0x443C00BCU

- #define **IOMUXC\_GPIO\_AON\_18\_CAN3\_TX** 0x443C0048U, 0x6U, 0, 0, 0x443C00BCU
- #define **IOMUXC\_GPIO\_AON\_19\_FLEXSPI2\_BUS2BIT\_B\_SCLK** 0x443C004CU, 0x0U, 0x42A1059CU, 0x1U, 0x443C00C0U
- #define **IOMUXC\_GPIO\_AON\_19\_LPSPI2\_SCK** 0x443C004CU, 0x1U, 0x443C0120U, 0x0U, 0x443C00C0U
- #define **IOMUXC\_GPIO\_AON\_19\_FLEXSPI2\_BUS2BIT\_A\_SS1\_B** 0x443C004CU, 0x3U, 0, 0, 0x443C00C0U
- #define **IOMUXC\_GPIO\_AON\_19\_LPUART1\_CTS\_B** 0x443C004CU, 0x4U, 0x443C0138U, 0x1U, 0x443C00C0U
- #define **IOMUXC\_GPIO\_AON\_19\_GPIO1\_IO19** 0x443C004CU, 0x5U, 0, 0, 0x443C00C0U
- #define **IOMUXC\_GPIO\_AON\_19\_CAN3\_RX** 0x443C004CU, 0x6U, 0x42A104A8U, 0x4U, 0x443C00C0U
- #define **IOMUXC\_GPIO\_AON\_19\_LPUART7 RTS\_B** 0x443C004CU, 0x8U, 0, 0, 0x443C00-C0U
- #define **IOMUXC\_GPIO\_AON\_19\_LPUART12\_TX** 0x443C004CU, 0x9U, 0x443C014CU, 0x1U, 0x443C00C0U
- #define **IOMUXC\_GPIO\_AON\_19\_ADC1\_CONV\_D00** 0x443C004CU, 0xCU, 0, 0, 0x443C00-C0U
- #define **IOMUXC\_GPIO\_AON\_20\_FLEXSPI2\_BUS2BIT\_B\_DQS** 0x443C0050U, 0x0U, 0x42A10574U, 0x2U, 0x443C00C4U
- #define **IOMUXC\_GPIO\_AON\_20\_FLEXSPI2\_BUS2BIT\_A\_SS1\_B** 0x443C0050U, 0x1U, 0, 0, 0x443C00C4U
- #define **IOMUXC\_GPIO\_AON\_20\_LPI2C1\_SDA** 0x443C0050U, 0x2U, 0x443C00F4U, 0x2U, 0x443C00C4U
- #define **IOMUXC\_GPIO\_AON\_20\_I3C1\_SDA** 0x443C0050U, 0x3U, 0x443C00ECU, 0x1U, 0x443C00C4U
- #define **IOMUXC\_GPIO\_AON\_20\_LPUART1\_RTS\_B** 0x443C0050U, 0x4U, 0, 0, 0x443C00-C4U
- #define **IOMUXC\_GPIO\_AON\_20\_GPIO1\_IO20** 0x443C0050U, 0x5U, 0, 0, 0x443C00C4U
- #define **IOMUXC\_GPIO\_AON\_20\_LPUART12\_RX** 0x443C0050U, 0x9U, 0x443C0148U, 0x1-U, 0x443C00C4U
- #define **IOMUXC\_GPIO\_AON\_20\_ADC1\_CONV\_D01** 0x443C0050U, 0xCU, 0, 0, 0x443C00-C4U
- #define **IOMUXC\_GPIO\_AON\_21\_ADC1\_CONV\_D02** 0x443C0054U, 0xCU, 0, 0, 0x443C00-C8U
- #define **IOMUXC\_GPIO\_AON\_21\_FLEXSPI2\_BUS2BIT\_B\_SS0\_B** 0x443C0054U, 0x0U, 0, 0, 0x443C00C8U
- #define **IOMUXC\_GPIO\_AON\_21\_LPSPI2\_PCS1** 0x443C0054U, 0x1U, 0x443C0118U, 0x1U, 0x443C00C8U
- #define **IOMUXC\_GPIO\_AON\_21\_LPI2C1\_SCL** 0x443C0054U, 0x2U, 0x443C00F0U, 0x2U, 0x443C00C8U
- #define **IOMUXC\_GPIO\_AON\_21\_I3C1\_SCL** 0x443C0054U, 0x3U, 0x443C00E8U, 0x1U, 0x443C00C8U
- #define **IOMUXC\_GPIO\_AON\_21\_SAI1\_TX\_DATA00** 0x443C0054U, 0x4U, 0, 0, 0x443C00-C8U
- #define **IOMUXC\_GPIO\_AON\_21\_GPIO1\_IO21** 0x443C0054U, 0x5U, 0, 0, 0x443C00C8U
- #define **IOMUXC\_GPIO\_AON\_21\_FLEXSPI2\_BUS2BIT\_A\_DQS** 0x443C0054U, 0x8U, 0x42A10570U, 0x1U, 0x443C00C8U
- #define **IOMUXC\_GPIO\_AON\_21\_SAI1\_RX\_DATA01** 0x443C0054U, 0x9U, 0x443C0174U, 0x1U, 0x443C00C8U

- #define **IOMUXC\_GPIO\_AON\_22\_CCMSRCGPC\_CCMOBS1** 0x443C0058U, 0xAU, 0, 0, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_ADC1\_CONV\_D03** 0x443C0058U, 0xCU, 0, 0, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_FLEXSPI2\_BUS2BIT\_A\_SS0\_B** 0x443C0058U, 0x0U, 0, 0, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_LPI2C2\_SDA** 0x443C0058U, 0x1U, 0x443C00FCU, 0x2U, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_LPUART7\_TX** 0x443C0058U, 0x2U, 0x443C0164U, 0x1U, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_LPUART12\_CTS\_B** 0x443C0058U, 0x3U, 0x443C0144U, 0x1U, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_SAI1\_TX\_SYNC** 0x443C0058U, 0x4U, 0x443C0180U, 0x1U, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_GPIO1\_IO22** 0x443C0058U, 0x5U, 0, 0, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_22\_LPSP12\_SCK** 0x443C0058U, 0x6U, 0x443C0120U, 0x1U, 0x443C00CCU
- #define **IOMUXC\_GPIO\_AON\_23\_FLEXSPI2\_BUS2BIT\_A\_SCLK** 0x443C005CU, 0x0U, 0x42A10598U, 0x1U, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_LPI2C2\_SCL** 0x443C005CU, 0x1U, 0x443C00F8U, 0x2U, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_LPUART7\_RX** 0x443C005CU, 0x2U, 0x443C0160U, 0x1U, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_LPUART12\_RTS\_B** 0x443C005CU, 0x3U, 0, 0, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_SAI1\_TX\_BCLK** 0x443C005CU, 0x4U, 0x443C017CU, 0x1U, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_GPIO1\_IO23** 0x443C005CU, 0x5U, 0, 0, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_LPSP12\_SDO** 0x443C005CU, 0x6U, 0x443C0128U, 0x2U, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_CCMSRCGPC\_CCMOBS2** 0x443C005CU, 0xAU, 0, 0, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_23\_ADC1\_CONV\_D04** 0x443C005CU, 0xCU, 0, 0, 0x443C00D0U
- #define **IOMUXC\_GPIO\_AON\_24\_ADC1\_CONV\_D05** 0x443C0060U, 0xCU, 0, 0, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_24\_FLEXSPI2\_BUS2BIT\_A\_DATA00** 0x443C0060U, 0x0U, 0x42A10578U, 0x1U, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_24\_LPI2C1\_SDA** 0x443C0060U, 0x1U, 0x443C00F4U, 0x3U, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_24\_LPUART2\_RTS\_B** 0x443C0060U, 0x2U, 0, 0, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_24\_LPUART7\_CTS\_B** 0x443C0060U, 0x3U, 0x443C015CU, 0x2U, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_24\_SAI1\_MCLK** 0x443C0060U, 0x4U, 0x443C0168U, 0x1U, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_24\_GPIO1\_IO24** 0x443C0060U, 0x5U, 0, 0, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_24\_LPSP12\_SDI** 0x443C0060U, 0x6U, 0x443C0124U, 0x2U, 0x443C00D4U
- #define **IOMUXC\_GPIO\_AON\_25\_ADC1\_CONV\_D06** 0x443C0064U, 0xCU, 0, 0, 0x443C00D4U

D8U

- #define **IOMUXC\_GPIO\_AON\_25\_FLEXSPI2\_BUS2BIT\_A\_DATA01** 0x443C0064U, 0x0U, 0x42A1057CU, 0x1U, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_25\_LPI2C1\_SCL** 0x443C0064U, 0x1U, 0x443C00F0U, 0x3U, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_25\_LPUART2\_CTS\_B** 0x443C0064U, 0x2U, 0x443C0150U, 0x1U, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_25\_LPUART7\_RTS\_B** 0x443C0064U, 0x3U, 0, 0, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_25\_SAI1\_RX\_DATA00** 0x443C0064U, 0x4U, 0x443C0170U, 0x1U, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_25\_GPIO1\_IO25** 0x443C0064U, 0x5U, 0, 0, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_25\_LPSPi2\_PCS0** 0x443C0064U, 0x6U, 0x443C0114U, 0x2U, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_25\_SAI1\_TX\_DATA01** 0x443C0064U, 0x7U, 0, 0, 0x443C00D8U
- #define **IOMUXC\_GPIO\_AON\_26\_FLEXSPI2\_BUS2BIT\_A\_DATA02** 0x443C0068U, 0x0U, 0x42A10580U, 0x1U, 0x443C00DCU
- #define **IOMUXC\_GPIO\_AON\_26\_LPSPi2\_PCS2** 0x443C0068U, 0x1U, 0, 0, 0x443C00DCU
- #define **IOMUXC\_GPIO\_AON\_26\_LPUART2\_TX** 0x443C0068U, 0x2U, 0x443C0158U, 0x1U, 0x443C00DCU
- #define **IOMUXC\_GPIO\_AON\_26\_SAI1\_RX\_BCLK** 0x443C0068U, 0x4U, 0x443C016CU, 0x1U, 0x443C00DCU
- #define **IOMUXC\_GPIO\_AON\_26\_GPIO1\_IO26** 0x443C0068U, 0x5U, 0, 0, 0x443C00DCU
- #define **IOMUXC\_GPIO\_AON\_26\_ADC1\_CONV\_D07** 0x443C0068U, 0xCU, 0, 0, 0x443C00DCU
- #define **IOMUXC\_GPIO\_AON\_27\_ADC1\_CONV\_RDY\_CLK** 0x443C006CU, 0xCU, 0, 0, 0x443C00E0U
- #define **IOMUXC\_GPIO\_AON\_27\_FLEXSPI2\_BUS2BIT\_A\_DATA03** 0x443C006CU, 0x0U, 0x42A10584U, 0x1U, 0x443C00E0U
- #define **IOMUXC\_GPIO\_AON\_27\_LPSPi2\_PCS3** 0x443C006CU, 0x1U, 0x443C011CU, 0x1U, 0x443C00E0U
- #define **IOMUXC\_GPIO\_AON\_27\_LPUART2\_RX** 0x443C006CU, 0x2U, 0x443C0154U, 0x1U, 0x443C00E0U
- #define **IOMUXC\_GPIO\_AON\_27\_SAI1\_RX\_SYNC** 0x443C006CU, 0x4U, 0x443C0178U, 0x1U, 0x443C00E0U
- #define **IOMUXC\_GPIO\_AON\_27\_GPIO1\_IO27** 0x443C006CU, 0x5U, 0, 0, 0x443C00E0U
- #define **IOMUXC\_GPIO\_AON\_27\_EWM\_EWM\_OUT\_B** 0x443C006CU, 0x7U, 0, 0, 0x443C00E0U
- #define **IOMUXC\_GPIO\_AON\_28\_DUMMY\_FLEXSPI2\_BUS2BIT\_A\_DQS** 0x443C0070U, 0x0U, 0x42A10570U, 0x2U, 0x443C00E4U
- #define **IOMUXC\_GPIO\_AON\_28\_DUMMY\_FLEXSPI2\_BUS2BIT\_B\_DQS** 0x443C0070U, 0x1U, 0x42A10574U, 0x3U, 0x443C00E4U
- #define **IOMUXC\_GPIO\_AON\_28\_DUMMY\_GPIO1\_IO28** 0x443C0070U, 0x5U, 0, 0, 0x443C00E4U

## Configuration

- static void **IOMUXC\_SetPinMux** (uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t inputOnfield)

- static void **IOMUXC\_SetPinConfig** (uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t configValue)
 

*Sets the IOMUXC pin mux mode.*

*Sets the IOMUXC pin configuration.*

## 7.2 Macro Definition Documentation

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

## 7.3 Function Documentation

### 7.3.1 static void IOMUXC\_SetPinMux ( *uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t inputOnfield* ) [inline], [static]

Note

The first five parameters can be filled with the pin function ID macros.

This is an example to set the PTA6 as the lpuart0\_tx:

```
* IOMUXC_SetPinMux(IOMUXC_PTA6_LPUART0_TX, 0);
*
```

This is an example to set the PTA0 as GPIOA0:

```
* IOMUXC_SetPinMux(IOMUXC_PTA0_GPIOA0, 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>   | Software input on field.   |

### 7.3.2 static void IOMUXC\_SetPinConfig ( *uint32\_t muxRegister, uint32\_t muxMode, uint32\_t inputRegister, uint32\_t inputDaisy, uint32\_t configRegister, uint32\_t configValue* ) [inline], [static]

## Note

The previous five parameters can be filled with the pin function ID macros.

This is an example to set pin configuration for IOMUXC\_PTA3\_LPI2C0\_SCLS:

```
* IOMUXC_SetPinConfig(IOMUXC_PTA3_LPI2C0_SCLS, IOMUXC_SW_PAD_CTL_PAD_PUS_MASK |
 IOMUXC_SW_PAD_CTL_PAD_PUS(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 8**

## **NIC301 Driver**

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

The NIC301 driver supports:

- Set IB configuration
- Get IB current configuration

# Chapter 9

## PMU Driver

### 9.1 Overview

The MCUXpresso SDK provides APIs for MCUXpresso SDK device's src operation.

The Power Management Unit(PMU) is designed to simplify the external power interface. Based on the features of PMU, the PMU driver is divided into 3 groups. In the function group "LDOs Control APIs", those APIs can be used to control LDOs (including PLL LDO, LPSR\_DIG LDO, LPSR\_ANA LDO) in software mode or hardware mode; In the function group "Bandgap Control APIs", those APIs can be used to control bandgap in software mode or hardware mode; In the function group "Body Bias Control APIs", those APIs can be used to control well bias and body bias in software mode or hardware mode.

### Data Structures

- struct [pmu\\_static\\_aon\\_ana\\_ldo\\_config\\_t](#)  
*AON ANA LDO config. [More...](#)*
- struct [pmu\\_static\\_aon\\_dig\\_config\\_t](#)  
*AON DIG LDO Config in Static/Software Mode. [More...](#)*
- struct [pmu\\_bbsm\\_dig\\_config\\_t](#)  
*BBSM DIG LDO config. [More...](#)*
- struct [pmu\\_static\\_bandgap\\_config\\_t](#)  
*Bandgap config in static mode. [More...](#)*
- union [pmu\\_well\\_bias\\_option\\_t](#)  
*The union of well bias basic options, such as clock source, power source and so on. [More...](#)*
- struct [pmu\\_well\\_bias\\_config\\_t](#)  
*The structure of well bias configuration. [More...](#)*

### Enumerations

- enum [pmu\\_ldo\\_name\\_t](#) {  
  kPMU\_PllLdo = 0U,  
  kPMU\_AonAnaLdo = 1U,  
  kPMU\_AonDigLdo = 2U }  
*The name of LDOs.*
- enum [pmu\\_control\\_mode\\_t](#) {  
  kPMU\_StaticMode = 0U,  
  kPMU\_GPCMode = 1U }  
*The control mode of LDOs/Bandgaps/Body Bias.*
- enum [pmu\\_ldo\\_operate\\_mode\\_t](#) {  
  kPMU\_LowPowerMode = 0x0U,  
  kPMU\_HighPowerMode = 0x1U }  
*The operation mode for the LDOs.*

- enum `pmu_aon_ana_ldo_charge_pump_current_t` {  
    `kPMU_AonAnaChargePump300nA` = 0U,  
    `kPMU_AonAnaChargePump400nA` = 1U,  
    `kPMU_AonAnaChargePump500nA` = 2U,  
    `kPMU_AonAnaChargePump600nA` = 3U }  
    *The enumeration of AON ANA LDO's charge pump current.*
- enum `pmu_aon_ana_ldo_output_range_t` {  
    `kPMU_AonAnaLdoOutputFrom1P77To1P83` = 0U,  
    `kPMU_AonAnaLdoOutputFrom1P72To1P77` = 1U,  
    `kPMU_AonAnaLdoOutputFrom1P82To1P88` = 2U }  
    *The enumeration of AON ANA LDO's output range.*
- enum `pmu_aon_dig_voltage_step_time_t` {  
    `kPMU_AonDigVoltageStepInc15us` = 0x0U,  
    `kPMU_AonDigVoltageStepInc25us` = 0x1U,  
    `kPMU_AonDigVoltageStepInc50us` = 0x2U,  
    `kPMU_AonDigVoltageStepInc100us` = 0x3U }  
    *The enumeration of voltage step time for AON DIG LDO.*
- enum `pmu_aon_dig_target_output_voltage_t` {

```

kPMU_AonDigTargetStableVoltage0P631V = 0x0U,
kPMU_AonDigTargetStableVoltage0P65V = 0x1U,
kPMU_AonDigTargetStableVoltage0P67V = 0x2U,
kPMU_AonDigTargetStableVoltage0P689V = 0x3U,
kPMU_AonDigTargetStableVoltage0P709V = 0x4U,
kPMU_AonDigTargetStableVoltage0P728V = 0x5U,
kPMU_AonDigTargetStableVoltage0P748V = 0x6U,
kPMU_AonDigTargetStableVoltage0P767V = 0x7U,
kPMU_AonDigTargetStableVoltage0P786V = 0x8U,
kPMU_AonDigTargetStableVoltage0P806V = 0x9U,
kPMU_AonDigTargetStableVoltage0P825V = 0xAU,
kPMU_AonDigTargetStableVoltage0P845V = 0xBU,
kPMU_AonDigTargetStableVoltage0P864V = 0xCU,
kPMU_AonDigTargetStableVoltage0P883V = 0xDU,
kPMU_AonDigTargetStableVoltage0P903V = 0xEU,
kPMU_AonDigTargetStableVoltage0P922V = 0xFU,
kPMU_AonDigTargetStableVoltage0P942V = 0x10U,
kPMU_AonDigTargetStableVoltage0P961V = 0x11U,
kPMU_AonDigTargetStableVoltage0P981V = 0x12U,
kPMU_AonDigTargetStableVoltage1P0V = 0x13U,
kPMU_AonDigTargetStableVoltage1P019V = 0x14U,
kPMU_AonDigTargetStableVoltage1P039V = 0x15U,
kPMU_AonDigTargetStableVoltage1P058V = 0x16U,
kPMU_AonDigTargetStableVoltage1P078V = 0x17U,
kPMU_AonDigTargetStableVoltage1P097V = 0x18U,
kPMU_AonDigTargetStableVoltage1P117V = 0x19U,
kPMU_AonDigTargetStableVoltage1P136V = 0x1AU,
kPMU_AonDigTargetStableVoltage1P155V = 0x1BU,
kPMU_AonDigTargetStableVoltage1P175V = 0x1CU,
kPMU_AonDigTargetStableVoltage1P194V = 0x1DU,
kPMU_AonDigTargetStableVoltage1P214V = 0x1EU,
kPMU_AonDigTargetStableVoltage1P233V = 0x1FU }

```

*The target output voltage of AON DIG LDO.*

- enum `pmu_bbsm_dig_charge_pump_current_t` {
   
kPMU\_BbsmDigChargePump12P5nA = 0U,
   
kPMU\_BbsmDigChargePump6P25nA = 1U,
   
kPMU\_BbsmDigChargePump18P75nA = 2U }

*The enumeration of the BBSM DIG LDO's charge pump current.*

- enum `pmu_bbsm_dig_discharge_resistor_value_t` {
   
kPMU\_BbsmDigDischargeResistor15K = 0U,
   
kPMU\_BbsmDigDischargeResistor30K = 1U,
   
kPMU\_BbsmDigDischargeResistor9K = 2U }

*The enumeration of the BBSM DIG LDO's discharge resistor.*

- enum `_pmu_static_bandgap_power_down_option` {

```
kPMU_PowerDownBandgapFully = 1U << 0U,
kPMU_PowerDownVoltageReferenceOutputOnly = 1U << 1U,
kPMU_PowerDownBandgapVBGUPDetector = 1U << 2U }
```

*The enumeration of bandgap power down option.*

- enum `pmu_bandgap_output_VBG_voltage_value_t` {
   
kPMU\_BandgapOutputVBGVoltageNominal = 0x0U,
   
kPMU\_BandgapOutputVBGVoltagePlus10mV = 0x1U,
   
kPMU\_BandgapOutputVBGVoltagePlus20mV = 0x2U,
   
kPMU\_BandgapOutputVBGVoltagePlus30mV = 0x3U,
   
kPMU\_BandgapOutputVBGVoltageMinus10mV = 0x4U,
   
kPMU\_BandgapOutputVBGVoltageMinus20mV = 0x5U,
   
kPMU\_BandgapOutputVBGVoltageMinus30mV = 0x6U,
   
kPMU\_BandgapOutputVBGVoltageMinus40mV = 0x7U }

*The enumeration of output VBG voltage.*

- enum `pmu_bandgap_output_current_value_t` {
   
kPMU\_OutputCurrent11P5uA = 0x0U,
   
kPMU\_OutputCurrent11P8uA = 0x1U,
   
kPMU\_OutputCurrent12P1uA = 0x2U,
   
kPMU\_OutputCurrent12P4uA = 0x4U,
   
kPMU\_OutputCurrent12P7uA = 0x5U,
   
kPMU\_OutputCurrent13P0uA = 0x6U,
   
kPMU\_OutputCurrent13P3uA = 0x7U }

*The enumeration of output current.*

- enum `pmu_well_bias_power_source_t` {
   
kPMU\_WellBiasPowerFromAonDigLdo = 0U,
   
kPMU\_WellBiasPowerFromDCDC }

*The enumerator of well bias power source.*

- enum `pmu_bias_area_size_t` {
   
kPMU\_180uA\_6mm2At125C = 0U,
   
kPMU\_150uA\_5mm2At125C,
   
kPMU\_120uA\_4mm2At125C,
   
kPMU\_90uA\_3mm2At125C,
   
kPMU\_60uA\_2mm2At125C,
   
kPMU\_45uA\_1P5mm2At125C,
   
kPMU\_30uA\_1mm2At125C,
   
kPMU\_15uA\_0P5mm2At125C }

*The enumerator of bias area size.*

- enum `pmu_well_bias_typical_freq_t` {
   
kPMU\_OscFreqDiv128 = 0U,
   
kPMU\_OscFreqDiv64 = 1U,
   
kPMU\_OscFreqDiv32 = 2U,
   
kPMU\_OscFreqDiv16 = 3U,
   
kPMU\_OscFreqDiv8 = 4U,
   
kPMU\_OscFreqDiv2 = 6U,
   
kPMU\_OscFreq = 7U }

*The enumerator of well bias typical frequency.*

- enum `pmu_adaptive_clock_source_t` {
   
    `kPMU_AdaptiveClkSourceOscClk` = 0U,
   
    `kPMU_AdaptiveClkSourceChargePumpClk` }
   
    *The enumerator of well bias adaptive clock source.*
- enum `pmu_freq_reduction_t` {
   
    `kPMU_FreqReductionNone` = 0U,
   
    `kPMU_FreqReduction30PCT`,
   
    `kPMU_FreqReduction40PCT`,
   
    `kPMU_FreqReduction50PCT` }
   
    *The enumerator of frequency reduction due to cap increment.*
- enum `pmu_well_bias_1P8_adjustment_t` {
   
    `kPMU_Cref0fFCspl0fFDeltaC0fF` = 0U,
   
    `kPMU_Cref0fFCspl30fFDeltaCN30fF`,
   
    `kPMU_Cref0fFCspl43fFDeltaCN43fF`,
   
    `kPMU_Cref0fFCspl62fFDeltaCN62fF`,
   
    `kPMU_Cref0fFCspl105fFDeltaCN105fF`,
   
    `kPMU_Cref30fFCspl0fFDeltaC30fF`,
   
    `kPMU_Cref30fFCspl43fFDeltaCN12fF`,
   
    `kPMU_Cref30fFCspl105fFDeltaCN75fF`,
   
    `kPMU_Cref43fFCspl0fFDeltaC43fF`,
   
    `kPMU_Cref43fFCspl30fFDeltaC13fF`,
   
    `kPMU_Cref43fFCspl62fFDeltaCN19fF`,
   
    `kPMU_Cref62fFCspl0fFDeltaC62fF`,
   
    `kPMU_Cref62fFCspl43fFDeltaC19fF`,
   
    `kPMU_Cref105fFCspl0fFDeltaC105fF`,
   
    `kPMU_Cref105fFCspl30fFDeltaC75fF` }
   
    *The enumerator of well bias 1P8 adjustment.*

## Driver version

- #define `FSL_PMU_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 2)`)
   
    *PMU driver version.*

## LDOs Control APIs

- void `PMU_SetPllLdoControlMode` (ANADIG\_PMU\_Type \*base, `pmu_control_mode_t` mode)
   
        *brief Selects the control mode of the PLL LDO.*
- void `PMU_StaticEnablePllLdo` (ANADIG\_PMU\_Type \*base)
   
        *Enables PLL LDO via AI interface in Static/Software mode.*
- void `PMU_StaticDisablePllLdo` (void)
   
        *Disables PLL LDO via AI interface in Static/Software mode.*
- void `PMU_StaticGetAonAnaLdoDefaultConfig` (`pmu_static_aon_ana_ldo_config_t` \*config)
   
        *Fill the AON ANA LDO configuration structure with default settings.*
- void `PMU_StaticAonAnaLdoInit` (ANADIG\_LDO\_BBSM\_Type \*base, const `pmu_static_aon_ana_ldo_config_t` \*config)
   
        *Initialize the AON ANA LDO in Static/Software Mode.*
- void `PMU_StaticAonAnaLdoDeinit` (ANADIG\_LDO\_BBSM\_Type \*base)
   
        *Disable the output of AON ANA LDO.*

- void **PMU\_StaticGetAonDigLdoDefaultConfig** (pmu\_static\_aon\_dig\_config\_t \*config)  
*Gets the default configuration of AON DIG LDO.*
- void **PMU\_StaticAonDigLdoInit** (ANADIG\_LDO\_BBSM\_Type \*base, const pmu\_static\_aon\_dig\_config\_t \*config)  
*Initialize the AON DIG LDO in static mode.*
- void **PMU\_StaticAonDigLdoDeinit** (ANADIG\_LDO\_BBSM\_Type \*base)  
*Disable the AON DIG LDO.*
- void **PMU\_GetBbsmDigLdoDefaultConfig** (pmu\_bbsm\_dig\_config\_t \*config)  
*Gets the default config of the BBSM DIG LDO.*
- void **PMU\_EnableLdoStandbyMode** (pmu\_ldo\_name\_t name, bool enable)  
*brief When STBY assert, enable/disable the selected LDO enter it's Low power mode.*

## Bandgap Control APIs

- void **PMU\_DisableBandgapSelfBiasAfterPowerUp** (void)  
*Disables Bandgap self bias for best noise performance.*
- void **PMU\_EnableBandgapSelfBiasBeforePowerDown** (void)  
*Enables Bandgap self bias before power down.*
- void **PMU\_StaticBandgapInit** (const pmu\_static\_bandgap\_config\_t \*config)  
*Initialize Bandgap.*

## Body Bias Control APIs

- void **PMU\_WellBiasInit** (ANADIG\_PMU\_Type \*base, const pmu\_well\_bias\_config\_t \*config)  
*Configures Well bias, such as power source, clock source and so on.*
- void **PMU\_GetWellBiasDefaultConfig** (pmu\_well\_bias\_config\_t \*config)  
*Gets the default configuration of well bias.*
- void **PMU\_EnableFBB** (ANADIG\_PMU\_Type \*base, bool enable)  
*brief Enables/disables FBB.*
- void **PMU\_EnableFBBStandbyMode** (ANADIG\_PMU\_Type \*base, bool enable)  
*brief Controls the ON/OFF of FBB when GPC send standby request.*

## 9.2 Data Structure Documentation

### 9.2.1 struct pmu\_static\_aon\_ana\_ldo\_config\_t

#### Data Fields

- pmu\_ldo\_operate\_mode\_t mode  
*The operate mode of AON ANA LDO.*
- bool enable2mALoad  
*Enable/Disable 2mA load.*
- bool enable4mALoad  
*Enable/Disable 4mA load.*
- bool enable20uALoad  
*Enable/Disable 20uA load.*
- bool enableStandbyMode  
*Enable/Disable Standby Mode.*

**Field Documentation**(1) **pmu\_ldo\_operate\_mode\_t pmu\_static\_aon\_ana\_ldo\_config\_t::mode**(2) **bool pmu\_static\_aon\_ana\_ldo\_config\_t::enable2mALoad**

- **true** Enables 2mA loading to prevent overshoot;
- **false** Disables 2mA loading.

(3) **bool pmu\_static\_aon\_ana\_ldo\_config\_t::enable4mALoad**

- **true** Enables 4mA loading to prevent dramatic voltage drop;
- **false** Disables 4mA load.

(4) **bool pmu\_static\_aon\_ana\_ldo\_config\_t::enable20uALoad**

- **true** Enables 20uA loading to prevent overshoot;
- **false** Disables 20uA load.

(5) **bool pmu\_static\_aon\_ana\_ldo\_config\_t::enableStandbyMode**

- **true** Enables Standby mode, if the STBY assert, the AON ANA LDO enter LP mode
- **false** Disables Standby mode.

**9.2.2 struct pmu\_static\_aon\_dig\_config\_t****Data Fields**

- **bool enableStableDetect**  
*Enable/Disable Stable Detect.*
- **pmu\_aon\_dig\_voltage\_step\_time\_t voltageStepTime**  
*Step time.*
- **pmu\_aon\_dig\_target\_output\_voltage\_t targetVoltage**  
*The target output voltage.*

**Field Documentation**(1) **bool pmu\_static\_aon\_dig\_config\_t::enableStableDetect**

- **true** Enables Stable Detect.
- **false** Disables Stable Detect.

- (2) `pmu_aon_dig_voltage_step_time_t pmu_static_aon_dig_config_t::voltageStepTime`
- (3) `pmu_aon_dig_target_output_voltage_t pmu_static_aon_dig_config_t::targetVoltage`

### 9.2.3 struct pmu\_bbsm\_dig\_config\_t

#### Data Fields

- `pmu_ldo_operate_mode_t mode`  
*The operate mode the BBSM DIG LDO.*
- `pmu_bbsm_dig_charge_pump_current_t chargePumpCurrent`  
*The current of BBSM DIG LDO's charge pump current.*
- `pmu_bbsm_dig_discharge_resistor_value_t dischargeResistorValue`  
*The value of BBSM DIG LDO's Discharge Resistor.*
- `uint8_t trimValue`  
*The trim value.*
- `bool enablePullDown`  
*Enable/Disable Pull down.*
- `bool enableLdoStable`  
*Enable/Disable BBSM DIG LDO Stable.*

#### Field Documentation

- (1) `pmu_ldo_operate_mode_t pmu_bbsm_dig_config_t::mode`
- (2) `pmu_bbsm_dig_charge_pump_current_t pmu_bbsm_dig_config_t::chargePumpCurrent`
- (3) `pmu_bbsm_dig_discharge_resistor_value_t pmu_bbsm_dig_config_t::dischargeResistorValue`
- (4) `uint8_t pmu_bbsm_dig_config_t::trimValue`
- (5) `bool pmu_bbsm_dig_config_t::enablePullDown`
  - **true** Enables the feature of using 1M ohm resistor to discharge the LDO output.
  - **false** Disables the feature of using 1M ohm resistor to discharge the LDO output.
- (6) `bool pmu_bbsm_dig_config_t::enableLdoStable`

### 9.2.4 struct pmu\_static\_bandgap\_config\_t

#### Data Fields

- `uint8_t powerDownOption`  
*The OR'ed value of `_pmu_static_bandgap_power_down_option`.*
- `bool enableLowPowerMode`  
*Turn on/off the Low power mode.*
- `pmu_bandgap_output_VBG_voltage_value_t outputVoltage`  
*The output VBG voltage of Bandgap.*
- `pmu_bandgap_output_current_value_t outputCurrent`

*The output current from the bandgap to the temperature sensors.*

### Field Documentation

(1) `uint8_t pmu_static_bandgap_config_t::powerDownOption`

Please refer to [\\_pmu\\_static\\_bandgap\\_power\\_down\\_option](#).

(2) `bool pmu_static_bandgap_config_t::enableLowPowerMode`

- **true** Turns on the low power operation of the bandgap.
- **false** Turns off the low power operation of the bandgap.

(3) `pmu_bandgap_output_VBG_voltage_value_t pmu_static_bandgap_config_t::outputVoltage`

(4) `pmu_bandgap_output_current_value_t pmu_static_bandgap_config_t::outputCurrent`

### 9.2.5 union pmu\_well\_bias\_option\_t

#### Data Fields

- `uint16_t wellBiasData`  
*well bias configuration data.*
- `uint16_t enablePWellOnly: 1U`  
*Turn on both PWELL and NWELL, or only trun on PWELL.*
- `uint16_t reserved1: 1U`  
*Reserved.*
- `uint16_t biasAreaSize: 3U`  
*Select size of bias area, please refer to [pmu\\_bias\\_area\\_size\\_t](#).*
- `uint16_t disableAdaptiveFreq: 1U`  
*Enable/Disable adaptive frequency.*
- `uint16_t wellBiasFreq: 3U`  
*Set well bias typical frequency, please refer to [pmu\\_well\\_bias\\_typical\\_freq\\_t](#).*
- `uint16_t clkSource: 1U`  
*Config the adaptive clock source, please [pmu\\_adaptive\\_clock\\_source\\_t](#).*
- `uint16_t freqReduction: 2U`  
*Config the percent of frequency reduction due to cap increment, please refer to [pmu\\_freq\\_reduction\\_t](#).*
- `uint16_t enablePullDownOption: 1U`  
*Enable/Disable pull down option.*
- `uint16_t reserved2: 1U`  
*Reserved.*
- `uint16_t powerSource: 1U`  
*Set power source, please refer to [pmu\\_well\\_bias\\_power\\_source\\_t](#).*
- `uint16_t reserved3: 1U`  
*Reserved.*

**Field Documentation**

- (1) `uint16_t pmu_well_bias_option_t::wellBiasData`
- (2) `uint16_t pmu_well_bias_option_t::enablePWellOnly`
  - **1b0** PWELL and NEWLL are both turned on.
  - **1b1** PWELL is turned on only.
- (3) `uint16_t pmu_well_bias_option_t::reserved1`
- (4) `uint16_t pmu_well_bias_option_t::disableAdaptiveFreq`
  - **1b0** Frequency change after each half cycle minimum frequency determined by typical frequency.
  - **1b1** Adaptive frequency disabled. Frequency determined by typical frequency.
- (5) `uint16_t pmu_well_bias_option_t::wellBiasFreq`
- (6) `uint16_t pmu_well_bias_option_t::clkSource`
- (7) `uint16_t pmu_well_bias_option_t::freqReduction`
- (8) `uint16_t pmu_well_bias_option_t::enablePullDownOption`
  - **false** Pull down option is disabled.
  - **true** Pull down option is enabled.
- (9) `uint16_t pmu_well_bias_option_t::reserved2`
- (10) `uint16_t pmu_well_bias_option_t::powerSource`
- (11) `uint16_t pmu_well_bias_option_t::reserved3`

**9.2.6 struct pmu\_well\_bias\_config\_t****Data Fields**

- `pmu_well_bias_option_t wellBiasOption`  
*Well bias basic function, please refer to [pmu\\_well\\_bias\\_option\\_t](#).*
- `pmu_well_bias_1P8_adjustment_t adjustment`  
*Well bias adjustment 1P8, please refer to [pmu\\_well\\_bias\\_1P8\\_adjustment\\_t](#).*

**Field Documentation**

- (1) `pmu_well_bias_option_t pmu_well_bias_config_t::wellBiasOption`
- (2) `pmu_well_bias_1P8_adjustment_t pmu_well_bias_config_t::adjustment`

## 9.3 Macro Definition Documentation

### 9.3.1 #define FSL\_PMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))

Version 2.1.2.

## 9.4 Enumeration Type Documentation

### 9.4.1 enum pmu\_ldo\_name\_t

Enumerator

*kPMU\_PlLdo* The PLL LDO in SOC domain.

*kPMU\_AonAnaLdo* The AON ANA LDO in AON domain.

*kPMU\_AonDigLdo* The AON DIG LDO in AON domain.

### 9.4.2 enum pmu\_control\_mode\_t

Enumerator

*kPMU\_StaticMode* Static/Software Control mode.

*kPMU\_GPCMode* GPC/Hardware Control mode.

### 9.4.3 enum pmu\_ldo\_operate\_mode\_t

Enumerator

*kPMU\_LowPowerMode* LDOs operate in Low power mode.

*kPMU\_HighPowerMode* LDOs operate in High power mode.

### 9.4.4 enum pmu\_aon\_ana\_ldo\_charge\_pump\_current\_t

Enumerator

*kPMU\_AonAnaChargePump300nA* The current of the charge pump is selected as 300nA.

*kPMU\_AonAnaChargePump400nA* The current of the charge pump is selected as 400nA.

*kPMU\_AonAnaChargePump500nA* The current of the charge pump is selected as 500nA.

*kPMU\_AonAnaChargePump600nA* The current of the charge pump is selected as 600nA.

### 9.4.5 enum pmu\_aon\_ana\_ldo\_output\_range\_t

Enumerator

- kPMU\_AonAnaLdoOutputFrom1P77To1P83* The output voltage varies from 1.77V to 1.83V.
- kPMU\_AonAnaLdoOutputFrom1P72To1P77* The output voltage varies from 1.72V to 1.77V.
- kPMU\_AonAnaLdoOutputFrom1P82To1P88* The output voltage varies from 1.82V to 1.88V.

### 9.4.6 enum pmu\_aon\_dig\_voltage\_step\_time\_t

Enumerator

- kPMU\_AonDigVoltageStepInc15us* AON DIG LDO voltage step time selected as 15us.
- kPMU\_AonDigVoltageStepInc25us* AON DIG LDO voltage step time selected as 25us.
- kPMU\_AonDigVoltageStepInc50us* AON DIG LDO voltage step time selected as 50us.
- kPMU\_AonDigVoltageStepInc100us* AON DIG LDO voltage step time selected as 100us.

### 9.4.7 enum pmu\_aon\_dig\_target\_output\_voltage\_t

Enumerator

- kPMU\_AonDigTargetStableVoltage0P631V* The target voltage selected as 0.631V.
- kPMU\_AonDigTargetStableVoltage0P65V* The target voltage selected as 0.65V.
- kPMU\_AonDigTargetStableVoltage0P67V* The target voltage selected as 0.67V.
- kPMU\_AonDigTargetStableVoltage0P689V* The target voltage selected as 0.689V.
- kPMU\_AonDigTargetStableVoltage0P709V* The target voltage selected as 0.709V.
- kPMU\_AonDigTargetStableVoltage0P728V* The target voltage selected as 0.728V.
- kPMU\_AonDigTargetStableVoltage0P748V* The target voltage selected as 0.748V.
- kPMU\_AonDigTargetStableVoltage0P767V* The target voltage selected as 0.767V.
- kPMU\_AonDigTargetStableVoltage0P786V* The target voltage selected as 0.786V.
- kPMU\_AonDigTargetStableVoltage0P806V* The target voltage selected as 0.806V.
- kPMU\_AonDigTargetStableVoltage0P825V* The target voltage selected as 0.825V.
- kPMU\_AonDigTargetStableVoltage0P845V* The target voltage selected as 0.845V.
- kPMU\_AonDigTargetStableVoltage0P864V* The target voltage selected as 0.864V.
- kPMU\_AonDigTargetStableVoltage0P883V* The target voltage selected as 0.883V.
- kPMU\_AonDigTargetStableVoltage0P903V* The target voltage selected as 0.903V.
- kPMU\_AonDigTargetStableVoltage0P922V* The target voltage selected as 0.922V.
- kPMU\_AonDigTargetStableVoltage0P942V* The target voltage selected as 0.942V.
- kPMU\_AonDigTargetStableVoltage0P961V* The target voltage selected as 0.961V.
- kPMU\_AonDigTargetStableVoltage0P981V* The target voltage selected as 0.981V.
- kPMU\_AonDigTargetStableVoltage1P0V* The target voltage selected as 1.0V.
- kPMU\_AonDigTargetStableVoltage1P019V* The target voltage selected as 1.019V.
- kPMU\_AonDigTargetStableVoltage1P039V* The target voltage selected as 1.039V.

|                                             |                                        |
|---------------------------------------------|----------------------------------------|
| <i>kPMU_AonDigTargetStableVoltage1P058V</i> | The target voltage selected as 1.058V. |
| <i>kPMU_AonDigTargetStableVoltage1P078V</i> | The target voltage selected as 1.078V. |
| <i>kPMU_AonDigTargetStableVoltage1P097V</i> | The target voltage selected as 1.097V. |
| <i>kPMU_AonDigTargetStableVoltage1P117V</i> | The target voltage selected as 1.117V. |
| <i>kPMU_AonDigTargetStableVoltage1P136V</i> | The target voltage selected as 1.136V. |
| <i>kPMU_AonDigTargetStableVoltage1P155V</i> | The target voltage selected as 1.155V. |
| <i>kPMU_AonDigTargetStableVoltage1P175V</i> | The target voltage selected as 1.175V. |
| <i>kPMU_AonDigTargetStableVoltage1P194V</i> | The target voltage selected as 1.194V. |
| <i>kPMU_AonDigTargetStableVoltage1P214V</i> | The target voltage selected as 1.214V. |
| <i>kPMU_AonDigTargetStableVoltage1P233V</i> | The target voltage selected as 1.233V. |

#### 9.4.8 enum pmu\_bbsm\_dig\_charge\_pump\_current\_t

Enumerator

|                                      |                                                                   |
|--------------------------------------|-------------------------------------------------------------------|
| <i>kPMU_BbsmDigChargePump12P5nA</i>  | The current of BBSM DIG LDO's charge pump is selected as 12.5nA.  |
| <i>kPMU_BbsmDigChargePump6P25nA</i>  | The current of BBSM DIG LDO's charge pump is selected as 6.25nA.  |
| <i>kPMU_BbsmDigChargePump18P75nA</i> | The current of BBSM DIG LDO's charge pump is selected as 18.75nA. |

#### 9.4.9 enum pmu\_bbsm\_dig\_discharge\_resistor\_value\_t

Enumerator

|                                         |                                                |
|-----------------------------------------|------------------------------------------------|
| <i>kPMU_BbsmDigDischargeResistor15K</i> | The Discharge Resistor is selected as 15K ohm. |
| <i>kPMU_BbsmDigDischargeResistor30K</i> | The Discharge Resistor is selected as 30K ohm. |
| <i>kPMU_BbsmDigDischargeResistor9K</i>  | The Discharge Resistor is selected as 9K ohm.  |

#### 9.4.10 enum \_pmu\_static\_bandgap\_power\_down\_option

Enumerator

|                                                 |                                                                                              |
|-------------------------------------------------|----------------------------------------------------------------------------------------------|
| <i>kPMU_PowerDownBandgapFully</i>               | Fully power down the bandgap module.                                                         |
| <i>kPMU_PowerDownVoltageReferenceOutputOnly</i> | Power down only the reference output section of the bandgap.                                 |
| <i>kPMU_PowerDownBandgapVBGUPDetector</i>       | Power down the VBGUP detector of the bandgap without affecting any additional functionality. |

**9.4.11 enum pmu\_bandgap\_output\_VBG\_voltage\_value\_t**

Enumerator

*kPMU\_BandgapOutputVBGVoltageNominal* Output nominal voltage.  
*kPMU\_BandgapOutputVBGVoltagePlus10mV* Output VBG voltage Plus 10mV.  
*kPMU\_BandgapOutputVBGVoltagePlus20mV* Output VBG voltage Plus 20mV.  
*kPMU\_BandgapOutputVBGVoltagePlus30mV* Output VBG voltage Plus 30mV.  
*kPMU\_BandgapOutputVBGVoltageMinus10mV* Output VBG voltage Minus 10mV.  
*kPMU\_BandgapOutputVBGVoltageMinus20mV* Output VBG voltage Minus 20mV.  
*kPMU\_BandgapOutputVBGVoltageMinus30mV* Output VBG voltage Minus 30mV.  
*kPMU\_BandgapOutputVBGVoltageMinus40mV* Output VBG voltage Minus 40mV.

**9.4.12 enum pmu\_bandgap\_output\_current\_value\_t**

Enumerator

*kPMU\_OutputCurrent11P5uA* Output 11.5uA current from the bandgap.  
*kPMU\_OutputCurrent11P8uA* Output 11.8uA current from the bandgap.  
*kPMU\_OutputCurrent12P1uA* Output 12.1uA current from the bandgap.  
*kPMU\_OutputCurrent12P4uA* Output 12.4uA current from the bandgap.  
*kPMU\_OutputCurrent12P7uA* Output 12.7uA current from the bandgap.  
*kPMU\_OutputCurrent13P0uA* Output 13.0uA current from the bandgap.  
*kPMU\_OutputCurrent13P3uA* Output 13.3uA current from the bandgap.

**9.4.13 enum pmu\_well\_bias\_power\_source\_t**

Enumerator

*kPMU\_WellBiasPowerFromAonDigLdo* AON Dig LDO supplies the power stage and NWELL sampler.  
*kPMU\_WellBiasPowerFromDCDC* DCDC supplies the power stage and NWELL sampler.

**9.4.14 enum pmu\_bias\_area\_size\_t**

Enumerator

*kPMU\_180uA\_6mm2At125C* Imax = 180uA; Areamax-RVT = 6.00mm<sup>2</sup> at 125C.  
*kPMU\_150uA\_5mm2At125C* Imax = 150uA; Areamax-RVT = 5.00mm<sup>2</sup> at 125C.  
*kPMU\_120uA\_4mm2At125C* Imax = 120uA; Areamax-RVT = 4.00mm<sup>2</sup> at 125C.  
*kPMU\_90uA\_3mm2At125C* Imax = 90uA; Areamax-RVT = 3.00mm<sup>2</sup> at 125C.  
*kPMU\_60uA\_2mm2At125C* Imax = 60uA; Areamax-RVT = 2.00mm<sup>2</sup> at 125C.

*kPMU\_45uA\_1P5mm2At125C* Imax = 45uA; Areamax-RVT = 1P5mm2 at 125C.

*kPMU\_30uA\_1mm2At125C* Imax = 30uA; Areamax-RVT = 1.00mm2 at 125C.

*kPMU\_15uA\_0P5mm2At125C* Imax = 15uA; Areamax-RVT = 0.50mm2 at 125C.

#### 9.4.15 enum pmu\_well\_bias\_typical\_freq\_t

Enumerator

*kPMU\_OscFreqDiv128* Typical frequency = osc\_freq / 128.

*kPMU\_OscFreqDiv64* Typical frequency = osc\_freq / 64.

*kPMU\_OscFreqDiv32* Typical frequency = osc\_freq / 32.

*kPMU\_OscFreqDiv16* Typical frequency = osc\_freq / 16.

*kPMU\_OscFreqDiv8* Typical frequency = osc\_freq / 8.

*kPMU\_OscFreqDiv2* Typical frequency = osc\_freq / 2.

*kPMU\_OscFreq* Typical frequency = oscillator frequency.

#### 9.4.16 enum pmu\_adaptive\_clock\_source\_t

Enumerator

*kPMU\_AdaptiveClkSourceOscClk* The adaptive clock source is oscillator clock.

*kPMU\_AdaptiveClkSourceChargePumpClk* The adaptive clock source is charge pump clock.

#### 9.4.17 enum pmu\_freq\_reduction\_t

Enumerator

*kPMU\_FreqReductionNone* No frequency reduction.

*kPMU\_FreqReduction30PCT* 30% frequency reduction due to cap increment.

*kPMU\_FreqReduction40PCT* 40% frequency reduction due to cap increment.

*kPMU\_FreqReduction50PCT* 50% frequency reduction due to cap increment.

#### 9.4.18 enum pmu\_well\_bias\_1P8\_adjustment\_t

Enumerator

*kPMU\_Cref0fFCspl0fFDeltaC0fF* Cref = 0fF, Cspl = 0fF, DeltaC = 0fF.

*kPMU\_Cref0fFCspl30fFDeltaCN30fF* Cref = 0fF, Cspl = 30fF, DeltaC = -30fF.

*kPMU\_Cref0fFCspl43fFDeltaCN43fF* Cref = 0fF, Cspl = 43fF, DeltaC = -43fF.

*kPMU\_Cref0fFCspl62fFDeltaCN62fF* Cref = 0fF, Cspl = 62fF, DeltaC = -62fF.

**kPMU\_Cref0fFCspl105fFDeltaCN105fF** Cref = 0fF, Cspl = 105fF, DeltaC = -105fF.  
**kPMU\_Cref30fFCspl0fFDeltaC30fF** Cref = 30fF, Cspl = 0fF, DeltaC = 30fF.  
**kPMU\_Cref30fFCspl43fFDeltaCN12fF** Cref = 30fF, Cspl = 43fF, DeltaC = -12fF.  
**kPMU\_Cref30fFCspl105fFDeltaCN75fF** Cref = 30fF, Cspl = 105fF, DeltaC = -75fF.  
**kPMU\_Cref43fFCspl0fFDeltaC43fF** Cref = 43fF, Cspl = 0fF, DeltaC = 43fF.  
**kPMU\_Cref43fFCspl30fFDeltaC13fF** Cref = 43fF, Cspl = 30fF, DeltaC = 13fF.  
**kPMU\_Cref43fFCspl62fFDeltaCN19fF** Cref = 43fF, Cspl = 62fF, DeltaC = -19fF.  
**kPMU\_Cref62fFCspl0fFDeltaC62fF** Cref = 62fF, Cspl = 0fF, DeltaC = 62fF.  
**kPMU\_Cref62fFCspl43fFDeltaC19fF** Cref = 62fF, Cspl = 43fF, DeltaC = 19fF.  
**kPMU\_Cref105fFCspl0fFDeltaC105fF** Cref = 105fF, Cspl = 0fF, DeltaC = 105fF.  
**kPMU\_Cref105fFCspl30fFDeltaC75fF** Cref = 105fF, Cspl = 30fF, DeltaC = 75fF.

## 9.5 Function Documentation

### 9.5.1 void PMU\_SetPIILdoControlMode ( ANADIG\_PMU\_Type \* *base*, pmu\_control\_mode\_t *mode* )

param base PMU peripheral base address. param mode The control mode of the PLL LDO. Please refer to pmu\_control\_mode\_t.

### 9.5.2 void PMU\_StaticEnablePIILdo ( ANADIG\_PMU\_Type \* *base* )

Parameters

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

### 9.5.3 void PMU\_StaticGetAonAnaLdoDefaultConfig ( pmu\_static\_aon\_ana\_ldo\_config\_t \* *config* )

The default values are:

```
* config->mode = kPMU_HighPowerMode;
 config->enable2mALoad = true;
 config->enable20uALoad = false;
 config->enable4mALoad = true;
 config->enableStandbyMode = false;
 config->driverStrength = kPMU_AonAnaLdoDriverStrength0;
 config->brownOutDetectorConfig = kPMU_AonAnaLdoBrownOutDetectorDisable;
 config->chargePumpCurrent = kPMU_AonAnaChargePump300nA;
 config->outputRange = kPMU_AonAnaLdoOutputFrom1P77To1P83
;
*
```

Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>config</i> | Pointer to the structure <a href="#">pmu_static_aon_ana_ldo_config_t</a> . |
|---------------|----------------------------------------------------------------------------|

#### 9.5.4 void PMU\_StaticAonAnaLdoInit ( ANADIG\_LDO\_BBSM\_Type \* *base*, const pmu\_static\_aon\_ana\_ldo\_config\_t \* *config* )

Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | ANADIG_LDO_BBSM peripheral base address.                                   |
| <i>config</i> | Pointer to the structure <a href="#">pmu_static_aon_ana_ldo_config_t</a> . |

#### 9.5.5 void PMU\_StaticAonAnaLdoDeinit ( ANADIG\_LDO\_BBSM\_Type \* *base* )

Parameters

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

#### 9.5.6 void PMU\_StaticGetAonDigLdoDefaultConfig ( pmu\_static\_aon\_dig\_config\_t \* *config* )

The default values are:

```
* config->enableStableDetect = false;
* config->voltageStepTime = kPMU_AonDigVoltageStepInc50us;
* config->brownOutConfig = kPMU_AonDigBrownOutDisable;
* config->targetVoltage = kPMU_AonDigTargetStableVoltage1P0V
;
* config->mode = kPMU_HighPowerMode;
```

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>config</i> | Pointer to the structure <a href="#">pmu_static_aon_dig_config_t</a> . |
|---------------|------------------------------------------------------------------------|

#### 9.5.7 void PMU\_StaticAonDigLdoInit ( ANADIG\_LDO\_BBSM\_Type \* *base*, const pmu\_static\_aon\_dig\_config\_t \* *config* )

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | ANADIG_LDO_BBSM peripheral base address.                               |
| <i>config</i> | Pointer to the structure <a href="#">pmu_static_aon_dig_config_t</a> . |

### 9.5.8 void PMU\_StaticAonDigLdoDeinit ( ANADIG\_LDO\_BBSM\_Type \* *base* )

Parameters

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

### 9.5.9 void PMU\_GetBbsmDigLdoDefaultConfig ( pmu\_bbsm\_dig\_config\_t \* *config* )

The default values are:

```
* config->mode = kPMU_LowPowerMode;
* config->chargePumpCurrent = kPMU_BbsmDigChargePump12P5nA;
* config->dischargeResistorValue = kPMU_BbsmDigDischargeResistor15K;
* config->trimValue = 0U;
* config->enablePullDown = true;
* config->enableLdoStable = false;
*
```

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>config</i> | Pointer to <a href="#">pmu_bbsm_dig_config_t</a> . |
|---------------|----------------------------------------------------|

### 9.5.10 void PMU\_EnableLdoStandbyMode ( pmu\_ldo\_name\_t *name*, bool *enable* )

param name The name of the selected ldo. Please see the enumeration pmu\_ldo\_name\_t for details. param enable Enable GPC standby mode or not.

### 9.5.11 void PMU\_DisableBandgapSelfBiasAfterPowerUp ( void )

This function should be invoked after powering up. This function will wait for the bandgap stable and disable the bandgap self bias. After powering up, it need to wait for the bandgap to get stable and then disable Bandgap Self bias for best noise performance.

**9.5.12 void PMU\_EnableBandgapSelfBiasBeforePowerDown ( void )**

This function will enable Bandgap self bias feature before powering down or there will be risk of Bandgap not starting properly.

**9.5.13 void PMU\_StaticBandgapInit ( const pmu\_static\_bandgap\_config\_t \* *config* )**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>config</i> | Pointer to the structure <a href="#">pmu_static_bandgap_config_t</a> . |
|---------------|------------------------------------------------------------------------|

**9.5.14 void PMU\_WellBiasInit ( ANADIG\_PMU\_Type \* *base*, const pmu\_well\_bias\_config\_t \* *config* )**

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | PMU peripheral base address.                                     |
| <i>config</i> | Pointer to the <a href="#">pmu_well_bias_config_t</a> structure. |

**9.5.15 void PMU\_GetWellBiasDefaultConfig ( pmu\_well\_bias\_config\_t \* *config* )**

Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>config</i> | The pointer to the <a href="#">pmu_well_bias_config_t</a> structure. |
|---------------|----------------------------------------------------------------------|

**9.5.16 void PMU\_EnableFBB ( ANADIG\_PMU\_Type \* *base*, bool *enable* )**

param base PMU peripheral base address. param enable Used to turn on/off FBB.

**9.5.17 void PMU\_EnableFBBStandbyMode ( ANADIG\_PMU\_Type \* *base*, bool *enable* )**

param base PMU peripheral base address. param enable Enable GPC standby mode or not.

# Chapter 10

## SOC\_SRC Driver

### 10.1 Overview

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

The System Reset Controller(SRC) is responsible for generation of all reset signals and the boot argument latching. To get the latched boot arguments, the driver provides "Latched Boot Arguments Related Interfaces" function group, all APIs in this group can be used to get the latched boot arguments. There are four situations cause an certain reset event:

- Power-on Reset
- Global System reset request
- Software configuration
- Low-power mode transaction The System Reset Controller(SRC) can generate reset events of the last 3 types. For the "Global System reset request", the APIs in "Global System Related Interfaces" can be used to set global reset mode or get the global reset status flags. For the "Software configuration", the APIs in "Slice Software Reset Related Interfaces" can be used. For the "Low-power mode transaction", the APIs in "Slice Low-Power Mode Transition Related Interfaces" can be used. If the reset behavior is controlled by CPU low-power mode, the function SRC\_SetSliceDomainModeAuthentication() and SRC\_SetSliceDomainModeConfig() can be invoked. If the the reset behavior is controlled by set point the function SRC\_SetSliceSetPointAuthentication() and SRC\_SetSliceSetPointConfig() can be used.

### Enumerations

- enum `src_core_name_t` {  
  `kSRC_CM33Core` = 0x1U,  
  `kSRC_CM7Core` = 0x2U }  
*System core.*
- enum `src_boot_fuse_selection_t` {  
  `kSRC_SerialDownloaderBootFlow` = 0U,  
  `kSRC_NormalBootFlow` = 1U }  
*The enumeration of the boot fuse selection.*
- enum `src_reset_source_t` {

```
kSRC_Wdog1Reset = 0U,
kSRC_Wdog2Reset = 1U,
kSRC_Wdog3Reset = 2U,
kSRC_Wdog4Reset = 3U,
kSRC_Wdog5Reset = 4U,
kSRC_TempsenseReset = 5U,
kSRC_SentinelReset = 6U,
kSRC_JtagSoftwareReset = 7U,
kSRC_CM33LockUpReset = 8U,
kSRC_CM33RequestReset = 9U,
kSRC_CM7LockUpReset = 10U,
kSRC_CM7RequestReset = 11U,
kSRC_DcdcOverVoltageReset = 12U,
kSRC_EcatResetOutputReset = 13U }
```

*The enumeration of global system reset sources.*

- enum `_src_reset_status_flags` {
 

```
kSRC_PORResetFlag = 1UL << 0UL,
kSRC_Wdog1ResetFlag = 1UL << 1UL,
kSRC_Wdog2ResetFlag = 1UL << 2UL,
kSRC_Wdog3ResetFlag = 1UL << 3UL,
kSRC_Wdog4ResetFlag = 1UL << 4UL,
kSRC_Wdog5ResetFlag = 1UL << 5UL,
kSRC_TempsenseResetFlag = 1UL << 6UL,
kSRC_SentinelResetFlag = 1UL << 7UL,
kSRC_JtagSoftwareResetFlag = 1UL << 8UL,
kSRC_CM33RequestResetFlag = 1UL << 9UL,
kSRC_CM33LockUpResetFlag = 1UL << 10UL,
kSRC_CM7RequestResetFlag = 1UL << 11UL,
kSRC_CM7LockUpResetFlag = 1UL << 12UL,
kSRC_DcdcOverVoltageResetFlag = 1UL << 13UL,
kSRC_EcatResetOutputResetFlag = 1UL << 14UL,
kSRC_IppResetFlag = 1UL << 16UL }
```

*The enumeration of reset status flags.*

- enum `src_reset_mode_t` {
 

```
kSRC_ResetSystem = 0x0U,
kSRC_DoNotResetSystem = 0x1U }
```

*The enumeration of global system reset mode.*

- enum `src_general_purpose_register_index_t` {

```
kSRC_GeneralPurposeRegister0 = 0U,
kSRC_GeneralPurposeRegister1,
kSRC_GeneralPurposeRegister2,
kSRC_GeneralPurposeRegister3,
kSRC_GeneralPurposeRegister4,
kSRC_GeneralPurposeRegister5,
kSRC_GeneralPurposeRegister6,
kSRC_GeneralPurposeRegister7,
kSRC_GeneralPurposeRegister8,
kSRC_GeneralPurposeRegister9,
kSRC_GeneralPurposeRegister10,
kSRC_GeneralPurposeRegister11,
kSRC_GeneralPurposeRegister12,
kSRC_GeneralPurposeRegister13,
kSRC_GeneralPurposeRegister14,
kSRC_GeneralPurposeRegister15,
kSRC_GeneralPurposeRegister16,
kSRC_GeneralPurposeRegister17,
kSRC_GeneralPurposeRegister18,
kSRC_GeneralPurposeRegister19 }
```

*The index of each general purpose register.*

- enum `src_power_ctrl_step_t` {
   
kSRC\_PDN\_EdgelockHandshake = 0UL,
   
kSRC\_PDN\_IsolationOn = 1UL,
   
kSRC\_PDN\_ResetAssert = 2UL,
   
kSRC\_PDN\_PowerSwitchOff = 3UL,
   
kSRC\_PUP\_PowerSwitchOn = 4UL,
   
kSRC\_PUP\_ResetRelease = 5UL,
   
kSRC\_PUP\_IsolationOff = 6UL,
   
kSRC\_PUP\_EdgeLockHandshake = 7UL }

*software power control step in power on/off sequence.*

- enum `src_power_level_t` {
   
kSRC\_PowerLevel2 = 0x2UL,
   
kSRC\_PowerLevel3 = 0x3UL,
   
kSRC\_PowerLevel4 = 0x4UL }

*The power level of low power mode setting.*

## Driver version

- #define `FSL_SRC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))  
*SRC driver version 2.0.1.*

## Core Reset release

- void `SRC_ReleaseCM7` (SRC\_GENERAL\_Type \*base)  
*Releases related core reset operation.*

## Latched Boot Arguments Related Interfaces

- static uint32\_t **SRC\_GetBootConfig** (SRC\_GENERAL\_Type \*base)  
*Gets Boot configuration.*
- static uint8\_t **SRC\_GetBootMode** (SRC\_GENERAL\_Type \*base)  
*Gets the latched state of the BOOT\_MODE1 and BOOT\_MODE0 signals.*

## Global System Related Interfaces

- void **SRC\_SetGlobalSystemResetMode** (SRC\_GENERAL\_Type \*base, **src\_reset\_source\_t** resetSource, **src\_reset\_mode\_t** resetMode)  
*Sets the reset mode of global system reset source.*
- static uint32\_t **SRC\_GetResetStatusFlags** (SRC\_GENERAL\_Type \*base)  
*Gets global system reset status flags.*
- static void **SRC\_ClearGlobalSystemResetStatus** (SRC\_GENERAL\_Type \*base, uint32\_t mask)  
*Clears the status of global reset.*

## General Purpose Registers Related Interfaces

- static void **SRC\_SetGeneralPurposeRegister** (SRC\_GENERAL\_Type \*base, **src\_general\_purpose\_register\_index\_t** index, uint32\_t value)  
*Sets value to general purpose registers.*
- static uint32\_t **SRC\_GetGeneralPurposeRegister** (SRC\_GENERAL\_Type \*base, **src\_general\_purpose\_register\_index\_t** index)  
*Gets the value from general purpose registers.*

## SRC Mix Slice Related Interfaces

- static void **SRC\_SLICE\_AllowUserModeAccess** (SRC\_MIX\_SLICE\_Type \*base, bool enable)  
*Allows/disallows user mode access.*
- static void **SRC\_SLICE\_AllowNonSecureModeAccess** (SRC\_MIX\_SLICE\_Type \*base, bool enable)  
*Allows/disallows non secure mode access.*
- static void **SRC\_SLICE\_LockAccessSetting** (SRC\_MIX\_SLICE\_Type \*base)  
*Locks the setting of user mode access and non secure mode access.*
- static void **SRC\_SLICE\_SetWhiteList** (SRC\_MIX\_SLICE\_Type \*base, uint8\_t domainId)  
*Sets the domain ID white list for the selected slice.*
- static void **SRC\_SLICE\_LockWhiteList** (SRC\_MIX\_SLICE\_Type \*base)  
*Locks the value of white list.*
- void **SRC\_SLICE\_SoftwareControl** (SRC\_MIX\_SLICE\_Type \*base, **src\_power\_ctrl\_step\_t** step)  
*brief Set software control step for slice power on/off sequence.*
- void **SRC\_SLICE\_PowerDown** (SRC\_MIX\_SLICE\_Type \*base, bool powerOff)  
*brief Power on/off slice.*
- void **SRC\_SLICE\_ControlByCpuLowPowerMode** (SRC\_MIX\_SLICE\_Type \*base, uint32\_t domainMap, **src\_power\_level\_t** level)

## 10.2 Macro Definition Documentation

### 10.2.1 #define FSL\_SRC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 10.3 Enumeration Type Documentation

### 10.3.1 enum src\_core\_name\_t

Enumerator

*kSRC\_CM33Core* System Core CM33.

*kSRC\_CM7Core* System Core CM7.

### 10.3.2 enum src\_boot\_fuse\_selection\_t

Enumerator

*kSRC\_SerialDownloaderBootFlow* The Boot flow jumps directly to the serial downloader.

*kSRC\_NormalBootFlow* The Boot flow follows the Normal Boot flow.

### 10.3.3 enum src\_reset\_source\_t

Enumerator

*kSRC\_Wdog1Reset* WDOG1 triggers the global system reset.

*kSRC\_Wdog2Reset* WDOG2 triggers the global system reset.

*kSRC\_Wdog3Reset* WDOG3 triggers the global system reset.

*kSRC\_Wdog4Reset* WODG4 triggers the global system reset.

*kSRC\_Wdog5Reset* WDOG5 triggers the global system reset.

*kSRC\_TempsenseReset* Tempsense triggers the global system reset.

*kSRC\_SentinelReset* Sentinel triggers the global system reset.

*kSRC\_JtagSoftwareReset* JTAG software triggers the global system reset.

*kSRC\_CM33LockUpReset* CM33 lockup triggers the global system reset.

*kSRC\_CM33RequestReset* CM33 request triggers the global system reset.

*kSRC\_CM7LockUpReset* CM7 lockup triggers the global system reset.

*kSRC\_CM7RequestReset* CM7 request triggers the global system reset.

*kSRC\_DcdcOverVoltageReset* DCDC over voltage triggers the global system reset.

*kSRC\_EcatResetOutputReset* EtherCAT reset output triggers the global system reset.

### 10.3.4 enum \_src\_reset\_status\_flags

Enumerator

*kSRC\_PORResetFlag* Reset is the result of POR.

*kSRC\_Wdog1ResetFlag* Reset is the result of watchdog1 time-out event.

*kSRC\_Wdog2ResetFlag* Reset is the result of watchdog2 time-out event.

*kSRC\_Wdog3ResetFlag* Reset is the result of watchdog3 time-out event.  
*kSRC\_Wdog4ResetFlag* Reset is the result of watchdog4 time-out event.  
*kSRC\_Wdog5ResetFlag* Reset is the result of watchdog5 time-out event.  
*kSRC\_TempsenseResetFlag* Reset is the result of software reset from temperature sensor.  
*kSRC\_SentinelResetFlag* Reset is the result of the Sentinel's reset event.  
*kSRC\_JtagSoftwareResetFlag* Reset is the result of software reset from JTAG.  
*kSRC\_CM33RequestResetFlag* Reset is the result of CM33 reset request.  
*kSRC\_CM33LockUpResetFlag* Reset is the result of CM33 CPU lockup.  
*kSRC\_CM7RequestResetFlag* Reset is the result of CM7 reset request.  
*kSRC\_CM7LockUpResetFlag* Reset is the result of CM7 CPU lockup.  
*kSRC\_DcdcOverVoltageResetFlag* Reset is the result of DCDC over voltage.  
*kSRC\_EcatResetOutputResetFlag* Reset is the result of ECAT reset output.  
*kSRC\_IppResetFlag* Reset is the result of chip PAD POR\_B.

### 10.3.5 enum src\_reset\_mode\_t

Enumerator

*kSRC\_ResetSystem* Generate the global system reset.  
*kSRC\_DoNotResetSystem* Do not generate the global system reset.

### 10.3.6 enum src\_general\_purpose\_register\_index\_t

Enumerator

*kSRC\_GeneralPurposeRegister0* The index of General Purpose Register0.  
*kSRC\_GeneralPurposeRegister1* The index of General Purpose Register1.  
*kSRC\_GeneralPurposeRegister2* The index of General Purpose Register2.  
*kSRC\_GeneralPurposeRegister3* The index of General Purpose Register3.  
*kSRC\_GeneralPurposeRegister4* The index of General Purpose Register4.  
*kSRC\_GeneralPurposeRegister5* The index of General Purpose Register5.  
*kSRC\_GeneralPurposeRegister6* The index of General Purpose Register6.  
*kSRC\_GeneralPurposeRegister7* The index of General Purpose Register7.  
*kSRC\_GeneralPurposeRegister8* The index of General Purpose Register8.  
*kSRC\_GeneralPurposeRegister9* The index of General Purpose Register9.  
*kSRC\_GeneralPurposeRegister10* The index of General Purpose Register10.  
*kSRC\_GeneralPurposeRegister11* The index of General Purpose Register11.  
*kSRC\_GeneralPurposeRegister12* The index of General Purpose Register12.  
*kSRC\_GeneralPurposeRegister13* The index of General Purpose Register13.  
*kSRC\_GeneralPurposeRegister14* The index of General Purpose Register14.  
*kSRC\_GeneralPurposeRegister15* The index of General Purpose Register15.  
*kSRC\_GeneralPurposeRegister16* The index of General Purpose Register16.  
*kSRC\_GeneralPurposeRegister17* The index of General Purpose Register17.

*kSRC\_GeneralPurposeRegister18* The index of General Purpose Register18.

*kSRC\_GeneralPurposeRegister19* The index of General Purpose Register19.

### 10.3.7 enum src\_power\_ctrl\_step\_t

Enumerator

*kSRC\_PDN\_EdgelockHandshake* Power down Edgelock handshake step.

*kSRC\_PDN\_IsolationOn* Power down isolation on step.

*kSRC\_PDN\_ResetAssert* Power down reset assert step.

*kSRC\_PDN\_PowerSwitchOff* Power down power switch off step.

*kSRC\_PUP\_PowerSwitchOn* Power up power switch on step.

*kSRC\_PUP\_ResetRelease* Power up reset release step.

*kSRC\_PUP\_IsolationOff* Power up isolation off step.

*kSRC\_PUP\_EdgeLockHandshake* Power up Edgelock handshake step.

### 10.3.8 enum src\_power\_level\_t

Enumerator

*kSRC\_PowerLevel2* power on when domain n is in RUN, off in WAIT/STOP/SUSPEND

*kSRC\_PowerLevel3* power on when domain n is in RUN/WAIT, off in STOP/SUSPEND

*kSRC\_PowerLevel4* power on when domain n is in RUN/WAIT/STOP, off in SUSPEND

## 10.4 Function Documentation

### 10.4.1 void SRC\_ReleaseCM7 ( SRC\_GENERAL\_Type \* *base* )

The core reset will be held until the boot core to release it.

Parameters

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

### 10.4.2 static uint32\_t SRC\_GetBootConfig ( SRC\_GENERAL\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Boot configuration. Please refer to fusemap.

#### 10.4.3 static uint8\_t SRC\_GetBootMode ( **SRC\_GENERAL\_Type** \* *base* ) [inline], [static]

Parameters

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

Returns

Boot mode. Please refer to the Boot mode pin setting section of System Boot.

#### 10.4.4 void SRC\_SetGlobalSystemResetMode ( **SRC\_GENERAL\_Type** \* *base*, *src\_reset\_source\_t* *resetSource*, *src\_reset\_mode\_t* *resetMode* )

This function sets the selected mode of the input global system reset sources.

Parameters

|                    |                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------|
| <i>base</i>        | SRC peripheral base address.                                                                |
| <i>resetSource</i> | The global system reset source. See <a href="#">src_reset_source_t</a> for more details.    |
| <i>resetMode</i>   | The reset mode of each reset source. See <a href="#">src_reset_mode_t</a> for more details. |

#### 10.4.5 static uint32\_t SRC\_GetResetStatusFlags ( **SRC\_GENERAL\_Type** \* *base* ) [inline], [static]

Parameters

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

Returns

The status of global system reset status. See [\\_src\\_reset\\_status\\_flags](#) for more details.

#### 10.4.6 static void SRC\_ClearGlobalSystemResetStatus ( SRC\_GENERAL\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SRC peripheral base address.                                                                                     |
| <i>mask</i> | The reset status flag to be cleared. See <a href="#">_src_global_system_reset_status_flags</a> for more details. |

#### 10.4.7 static void SRC\_SetGeneralPurposeRegister ( SRC\_GENERAL\_Type \* *base*, src\_general\_purpose\_register\_index\_t *index*, uint32\_t *value* ) [inline], [static]

Parameters

|              |                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i>  | SRC peripheral base address.                                                                             |
| <i>index</i> | The index of GPRx register array. Please refer to <a href="#">src_general_purpose_register_index_t</a> . |
| <i>value</i> | Setting value for GPRx register.                                                                         |

#### 10.4.8 static uint32\_t SRC\_GetGeneralPurposeRegister ( SRC\_GENERAL\_Type \* *base*, src\_general\_purpose\_register\_index\_t *index* ) [inline], [static]

Parameters

|              |                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i>  | SRC peripheral base address.                                                                             |
| <i>index</i> | The index of GPRx register array. Please refer to <a href="#">src_general_purpose_register_index_t</a> . |

Returns

The setting value for GPRx register.

#### 10.4.9 static void SRC\_SLICE\_AllowUserModeAccess ( SRC\_MIX\_SLICE\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                                                                                                                              |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SRC peripheral base address.                                                                                                                                                 |
| <i>enable</i> | Used to control user mode access. <ul style="list-style-type: none"> <li>• <b>true</b> Allow user mode access.</li> <li>• <b>false</b> Disallow user mode access.</li> </ul> |

#### 10.4.10 static void SRC\_SLICE\_AllowNonSecureModeAccess ( SRC\_MIX\_SLICE\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                                                                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SRC peripheral base address.                                                                                                                                                                   |
| <i>enable</i> | Used to control non secure mode access. <ul style="list-style-type: none"> <li>• <b>true</b> Allow non secure mode access.</li> <li>• <b>false</b> Disallow non secure mode access.</li> </ul> |

#### 10.4.11 static void SRC\_SLICE\_LockAccessSetting ( SRC\_MIX\_SLICE\_Type \* *base* ) [inline], [static]

Note

Once locked only reset can unlock related settings.

Parameters

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

**10.4.12 static void SRC\_SLICE\_SetWhiteList ( SRC\_MIX\_SLICE\_Type \* *base*, uint8\_t *domainId* ) [inline], [static]**

Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>     | SRC peripheral base address.                                                                 |
| <i>domainId</i> | The core to access registers, should be the OR'ed value of <a href="#">src_core_name_t</a> . |

**10.4.13 static void SRC\_SLICE\_LockWhiteList ( SRC\_MIX\_SLICE\_Type \* *base* ) [inline], [static]**

Note

Once locked only reset can unlock related settings.

Parameters

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

**10.4.14 void SRC\_SLICE\_SoftwareControl ( SRC\_MIX\_SLICE\_Type \* *base*, src\_power\_ctrl\_step\_t *step* )**

param base SRC peripheral base address. param step Slice power on/off sequence step. See [src\\_power\\_ctrl\\_step\\_t](#) for more details.

**10.4.15 void SRC\_SLICE\_PowerDown ( SRC\_MIX\_SLICE\_Type \* *base*, bool *powerOff* )**

param base SRC peripheral base address. param powerOff Used to trigger slice power on/off sequence.

- **true** Trigger a power off sequence.
- **false** Trigger a power on sequence.

# Chapter 11

## ASRC: Asynchronous sample rate converter

### 11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Asynchronous sample rate converter module of MCUXpresso SDK devices.

The ASRC supports up to three sampling rate pairs, the ASRC supports concurrent sample rate conversion of up to 10 channels. The incoming audio data to this chip may be received from various sources at different sampling rates. The outgoing audio data of this chip may have different sampling rates and it can also be associated with output clocks that are asynchronous to the input clocks. When the input sampling clock is not physically available, the rate conversion can still work by setting ideal-ratio values into ASRC interface registers. When both the input sampling clock and the output sampling clock are physically available, the rate conversion can work by configuring the physical clocks.

The Asynchronous sample rate converter support convert between sample rate: kASRC\_SampleRate\_8000HZ = 8000U, /\*!< asrc sample rate 8KHZ

### Modules

- [ASRC Driver](#)

## 11.2 ASRC Driver

### 11.2.1 Overview

#### Data Structures

- struct `asrc_channel_pair_config_t`  
*asrc channel pair configuration* [More...](#)
- struct `asrc_transfer_t`  
*SAI transfer structure.* [More...](#)
- struct `asrc_in_handle_t`  
*asrc in handler* [More...](#)
- struct `asrc_out_handle_t`  
*output handler* [More...](#)
- struct `asrc_handle_t`  
*ASRC handle structure.* [More...](#)

#### Macros

- `#define ASRC_XFER_QUEUE_SIZE (4U)`  
*ASRC transfer queue size, user can refine it according to use case.*
- `#define FSL_ASRC_CHANNEL_PAIR_COUNT (4U)`  
*ASRC channel pair count.*
- `#define FSL_ASRC_CHANNEL_PAIR_FIFO_DEPTH (64U)`  
*ASRC FIFO depth.*
- `#define ASRC_ASRCTR_AT_MASK(index) ((uint32_t)1U << (ASRC_ASRCTR_ATSA_SHIFT + (uint32_t)(index)))`  
*ASRC register access macro.*

#### Typedefs

- `typedef void(* asrc_transfer_callback_t )(ASRC_Type *base, asrc_handle_t *handle, status_t status, void *userData)`  
*ASRC transfer callback prototype.*

## Enumerations

- enum {
   
*kStatus\_ASRCIdle = MAKE\_STATUS(kStatusGroup\_ASRC, 0),*  
*kStatus\_ASRCInIdle = MAKE\_STATUS(kStatusGroup\_ASRC, 1),*  
*kStatus\_ASRCOutIdle = MAKE\_STATUS(kStatusGroup\_ASRC, 2),*  
*kStatus\_ASRCBusy = MAKE\_STATUS(kStatusGroup\_ASRC, 3),*  
*kStatus\_ASRCInvalidArgument = MAKE\_STATUS(kStatusGroup\_ASRC, 4),*  
*kStatus\_ASRCClockConfigureFailed = MAKE\_STATUS(kStatusGroup\_ASRC, 5),*  
*kStatus\_ASRCChannelPairConfigureFailed = MAKE\_STATUS(kStatusGroup\_ASRC, 6),*  
*kStatus\_ASRCConvertError = MAKE\_STATUS(kStatusGroup\_ASRC, 7),*  
*kStatus\_ASRCNotSupport = MAKE\_STATUS(kStatusGroup\_ASRC, 8),*  
*kStatus\_ASRCQueueFull = MAKE\_STATUS(kStatusGroup\_ASRC, 9),*  
*kStatus\_ASRCOutQueueIdle = MAKE\_STATUS(kStatusGroup\_ASRC, 10),*  
*kStatus\_ASRCInQueueIdle = MAKE\_STATUS(kStatusGroup\_ASRC, 11) }*  
*ASRC return status.*
- enum **asrc\_channel\_pair\_t** {
   
*kASRC\_ChannelPairA = 0,*  
*kASRC\_ChannelPairB = 1,*  
*kASRC\_ChannelPairC = 2 }*  
*ASRC channel pair mask.*
- enum {
   
*kASRC\_SampleRate\_8000HZ = 8000U,*  
*kASRC\_SampleRate\_11025HZ = 11025U,*  
*kASRC\_SampleRate\_12000HZ = 12000U,*  
*kASRC\_SampleRate\_16000HZ = 16000U,*  
*kASRC\_SampleRate\_22050HZ = 22050U,*  
*kASRC\_SampleRate\_24000HZ = 24000U,*  
*kASRC\_SampleRate\_30000HZ = 30000U,*  
*kASRC\_SampleRate\_32000HZ = 32000U,*  
*kASRC\_SampleRate\_44100HZ = 44100U,*  
*kASRC\_SampleRate\_48000HZ = 48000U,*  
*kASRC\_SampleRate\_64000HZ = 64000U,*  
*kASRC\_SampleRate\_88200HZ = 88200U,*  
*kASRC\_SampleRate\_96000HZ = 96000U,*  
*kASRC\_SampleRate\_128000HZ = 128000U,*  
*kASRC\_SampleRate\_176400HZ = 176400U,*  
*kASRC\_SampleRate\_192000HZ = 192000U }*  
*ASRC support sample rate.*
- enum {

```

kASRC_FPIInWaitStateInterruptEnable = ASRC_ASRIER_AFPWE_MASK,
kASRC_OverLoadInterruptMask = ASRC_ASRIER_AOLIE_MASK,
kASRC_DataOutputCInterruptMask = ASRC_ASRIER_ADOEC_MASK,
kASRC_DataOutputBInterruptMask = ASRC_ASRIER_ADOEB_MASK,
kASRC_DataOutputAInterruptMask = ASRC_ASRIER_ADOEA_MASK,
kASRC_DataInputCInterruptMask = ASRC_ASRIER_ADIEC_MASK,
kASRC_DataInputBInterruptMask = ASRC_ASRIER_ADIEB_MASK,
kASRC_DataInputAInterruptMask = ASRC_ASRIER_ADIEA_MASK }

```

*The ASRC interrupt enable flag.*

- enum {

```

kASRC_StatusDSLCounterReady = ASRC_ASRSTR_DSLCNT_MASK,
kASRC_StatusTaskQueueOverLoad = ASRC_ASRSTR_ATQOL_MASK,
kASRC_StatusPairCOutputOverLoad = ASRC_ASRSTR_AOOLC_MASK,
kASRC_StatusPairBOutputOverLoad = ASRC_ASRSTR_AOOLB_MASK,
kASRC_StatusPairAOutputOverLoad = ASRC_ASRSTR_AOOLA_MASK,
kASRC_StatusPairCInputOverLoad = ASRC_ASRSTR_AIOLC_MASK,
kASRC_StatusPairBInputOverLoad = ASRC_ASRSTR_AIOLB_MASK,
kASRC_StatusPairAInputOverLoad = ASRC_ASRSTR_AIOLA_MASK,
kASRC_StatusPairCOutputOverflow = ASRC_ASRSTR_AODOC_MASK,
kASRC_StatusPairBOutputOverflow = ASRC_ASRSTR_AODOB_MASK,
kASRC_StatusPairAOutputOverflow = ASRC_ASRSTR_AODOA_MASK,
kASRC_StatusPairCInputUnderflow = ASRC_ASRSTR_AIDUC_MASK,
kASRC_StatusPairBInputUnderflow = ASRC_ASRSTR_AIDUB_MASK,
kASRC_StatusPairAInputUnderflow = ASRC_ASRSTR_AIDUA_MASK,
kASRC_StatusFPIInWaitState = ASRC_ASRSTR_FPWT_MASK,
kASRC_StatusOverloadError = ASRC_ASRSTR_AOLE_MASK,
kASRC_StatusInputError,
kASRC_StatusOutputError,
kASRC_StatusPairCOutputReady = ASRC_ASRSTR_AODFC_MASK,
kASRC_StatusPairBOutputReady = ASRC_ASRSTR_AODFB_MASK,
kASRC_StatusPairAOutputReady = ASRC_ASRSTR_AODFA_MASK,
kASRC_StatusPairCInputReady = ASRC_ASRSTR_AIDEC_MASK,
kASRC_StatusPairBInputReady = ASRC_ASRSTR_AIDEB_MASK,
kASRC_StatusPairAInputReady = ASRC_ASRSTR_AIDEA_MASK,
kASRC_StatusPairAInterrupt = kASRC_StatusPairAInputReady | kASRC_StatusPairAOutputReady,
kASRC_StatusPairBInterrupt = kASRC_StatusPairBInputReady | kASRC_StatusPairBOutputReady,
kASRC_StatusPairCInterrupt = kASRC_StatusPairCInputReady | kASRC_StatusPairCOutputReady
 }
```

*The ASRC interrupt status.*

- enum {

```

kASRC_OutputFifoNearFull = ASRC_ASRFSTA_OAFA_MASK,
kASRC_InputFifoNearEmpty = ASRC_ASRFSTA_IAEA_MASK }

```

*ASRC channel pair status.*

- enum `asrc_ratio_t` {
   
    `kASRC_RatioNotUsed` = 0U,
   
    `kASRC_RatioUseInternalMeasured`,
   
    `kASRC_RatioUseIdealRatio` }
   
        *ASRC ideal ratio.*
- enum `asrc_audio_channel_t` {
   
    `kASRC_ChannelsNumber1` = 1U,
   
    `kASRC_ChannelsNumber2` = 2U,
   
    `kASRC_ChannelsNumber3` = 3U,
   
    `kASRC_ChannelsNumber4` = 4U,
   
    `kASRC_ChannelsNumber5` = 5U,
   
    `kASRC_ChannelsNumber6` = 6U,
   
    `kASRC_ChannelsNumber7` = 7U,
   
    `kASRC_ChannelsNumber8` = 8U,
   
    `kASRC_ChannelsNumber9` = 9U,
   
    `kASRC_ChannelsNumber10` = 10U }
   
        *Number of channels in audio data.*
- enum `asrc_data_width_t` {
   
    `kASRC_DataWidth24Bit` = 0U,
   
    `kASRC_DataWidth16Bit` = 1U,
   
    `kASRC_DataWidth8Bit` = 2U }
   
        *data width*
- enum `asrc_data_align_t` {
   
    `kASRC_DataAlignMSB` = 1U,
   
    `kASRC_DataAlignLSB` = 0U }
   
        *data alignment*
- enum `asrc_sign_extension_t` {
   
    `kASRC_NoSignExtension` = 0U,
   
    `kASRC_SignExtension` = 1U }
   
        *sign extension*

## Driver version

- #define `FSL_ASRC_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 3)`)
   
        *Version 2.1.3.*

## Initialization and deinitialization

- `uint32_t ASRC_GetInstance (ASRC_Type *base)`
  
          *Get instance number of the ASRC peripheral.*
- `void ASRC_Init (ASRC_Type *base, uint32_t asrcPeripheralClock_Hz)`
  
          *brief Initializes the asrc peripheral.*
- `void ASRC_Deinit (ASRC_Type *base)`
  
          *De-initializes the ASRC peripheral.*
- `void ASRC_SoftwareReset (ASRC_Type *base)`

- *Do software reset.*
- `status_t ASRC_SetChannelPairConfig` (`ASRC_Type *base, asrc_channel_pair_t channelPair, asrc_channel_pair_config_t *config, uint32_t inputSampleRate, uint32_t outputSampleRate)`  
*ASRC configure channel pair.*
- `uint32_t ASRC_GetOutSamplesSize` (`ASRC_Type *base, asrc_channel_pair_t channelPair, uint32_t inSampleRate, uint32_t outSampleRate, uint32_t inSamplesize)`  
*Get output sample buffer size.*
- `uint32_t ASRC_MapSamplesWidth` (`ASRC_Type *base, asrc_channel_pair_t channelPair, uint32_t *inWidth, uint32_t *outWidth)`  
*Map register sample width to real sample width.*
- `uint32_t ASRC_GetRemainFifoSamples` (`ASRC_Type *base, asrc_channel_pair_t channelPair, uint32_t *buffer, uint32_t outSampleWidth, uint32_t remainSamples)`  
*Get left samples in fifo.*
- `static void ASRC_ModuleEnable` (`ASRC_Type *base, bool enable)`  
*ASRC module enable.*
- `static void ASRC_ChannelPairEnable` (`ASRC_Type *base, asrc_channel_pair_t channelPair, bool enable)`  
*ASRC enable channel pair.*

## Interrupts

- `static void ASRC_EnableInterrupt` (`ASRC_Type *base, uint32_t mask)`  
*ASRC interrupt enable This function enable the ASRC interrupt with the provided mask.*
- `static void ASRC_DisableInterrupt` (`ASRC_Type *base, uint32_t mask)`  
*ASRC interrupt disable This function disable the ASRC interrupt with the provided mask.*

## Status

- `static uint32_t ASRC_GetStatus` (`ASRC_Type *base)`  
*Gets the ASRC status flag state.*
- `static bool ASRC_GetChannelPairInitialStatus` (`ASRC_Type *base, asrc_channel_pair_t channel)`  
*Gets the ASRC channel pair initialization state.*
- `static uint32_t ASRC_GetChannelPairFifoStatus` (`ASRC_Type *base, asrc_channel_pair_t channelPair)`  
*Gets the ASRC channel A fifo a status flag state.*

## Bus Operations

- `static void ASRC_ChannelPairWriteData` (`ASRC_Type *base, asrc_channel_pair_t channelPair, uint32_t data)`  
*Writes data into ASRC channel pair FIFO.*
- `static uint32_t ASRC_ChannelPairReadData` (`ASRC_Type *base, asrc_channel_pair_t channelPair)`  
*Read data from ASRC channel pair FIFO.*
- `static uint32_t ASRC_GetInputDataRegisterAddress` (`ASRC_Type *base, asrc_channel_pair_t channelPair)`  
*Get input data fifo address.*

- static uint32\_t [ASRC\\_GetOutputDataRegisterAddress](#) (ASRC\_Type \*base, [asrc\\_channel\\_pair\\_t](#) channelPair)  
*Get output data fifo address.*
- [status\\_t ASRC\\_SetIdealRatioConfig](#) (ASRC\_Type \*base, [asrc\\_channel\\_pair\\_t](#) channelPair, uint32\_t inputSampleRate, uint32\_t outputSampleRate)  
*ASRC configure ideal ratio.*

## Transactional

- [status\\_t ASRC\\_TransferSetChannelPairConfig](#) (ASRC\_Type \*base, [asrc\\_handle\\_t](#) \*handle, [asrc\\_channel\\_pair\\_config\\_t](#) \*config, uint32\_t inputSampleRate, uint32\_t outputSampleRate)  
*ASRC configure channel pair.*
- void [ASRC\\_TransferCreateHandle](#) (ASRC\_Type \*base, [asrc\\_handle\\_t](#) \*handle, [asrc\\_channel\\_pair\\_t](#) channelPair, [asrc\\_transfer\\_callback\\_t](#) inCallback, [asrc\\_transfer\\_callback\\_t](#) outCallback, void \*userData)  
*Initializes the ASRC handle.*
- [status\\_t ASRC\\_TransferNonBlocking](#) (ASRC\_Type \*base, [asrc\\_handle\\_t](#) \*handle, [asrc\\_transfer\\_t](#) \*xfer)  
*Performs an interrupt non-blocking convert on asrc.*
- [status\\_t ASRC\\_TransferBlocking](#) (ASRC\_Type \*base, [asrc\\_channel\\_pair\\_t](#) channelPair, [asrc\\_transfer\\_t](#) \*xfer)  
*Performs an blocking convert on asrc.*
- [status\\_t ASRC\\_TransferGetConvertedCount](#) (ASRC\_Type \*base, [asrc\\_handle\\_t](#) \*handle, size\_t \*count)  
*Get converted byte count.*
- void [ASRC\\_TransferAbortConvert](#) (ASRC\_Type \*base, [asrc\\_handle\\_t](#) \*handle)  
*Aborts the current convert.*
- void [ASRC\\_TransferTerminateConvert](#) (ASRC\_Type \*base, [asrc\\_handle\\_t](#) \*handle)  
*Terminate all ASRC convert.*
- void [ASRC\\_TransferHandleIRQ](#) (ASRC\_Type \*base, [asrc\\_handle\\_t](#) \*handle)  
*ASRC convert interrupt handler.*

## 11.2.2 Data Structure Documentation

### 11.2.2.1 struct [asrc\\_channel\\_pair\\_config\\_t](#)

#### Data Fields

- [asrc\\_audio\\_channel\\_t](#) audioDataChannels  
*audio data channel numbers*
- [asrc\\_clock\\_source\\_t](#) inClockSource  
*input clock source, reference the clock source definition in SOC header file*
- uint32\_t inSourceClock\_Hz  
*input source clock frequency*
- [asrc\\_clock\\_source\\_t](#) outClockSource  
*output clock source, reference the clock source definition in SOC header file*
- uint32\_t outSourceClock\_Hz

- *output source clock frequency*
- **asrc\_ratio\_t sampleRateRatio**  
*sample rate ratio type*
- **asrc\_data\_width\_t inDataWidth**  
*input data width*
- **asrc\_data\_align\_t inDataAlign**  
*input data alignment*
- **asrc\_data\_width\_t outDataWidth**  
*output data width*
- **asrc\_data\_align\_t outDataAlign**  
*output data alignment*
- **asrc\_sign\_extension\_t outSignExtension**  
*output extension*
- **uint8\_t outFifoThreshold**  
*output fifo threshold*
- **uint8\_t inFifoThreshold**  
*input fifo threshold*
- **bool bufStallWhenFifoEmptyFull**  
*stall Pair A conversion in case of Buffer near empty full condition*

### 11.2.2.2 struct asrc\_transfer\_t

#### Data Fields

- **void \* inData**  
*Data address to convert.*
- **size\_t inDataSize**  
*input data size.*
- **void \* outData**  
*Data address to store converted data.*
- **size\_t outDataSize**  
*output data size.*

#### Field Documentation

- (1) **void\* asrc\_transfer\_t::inData**
- (2) **size\_t asrc\_transfer\_t::inDataSize**
- (3) **size\_t asrc\_transfer\_t::outDataSize**

### 11.2.2.3 struct asrc\_in\_handle\_t

#### Data Fields

- **asrc\_transfer\_callback\_t callback**  
*Callback function called at convert complete.*
- **uint32\_t sampleWidth**  
*data width*
- **uint32\_t sampleMask**

- *data mask*
- `uint32_t fifoThreshold`  
*fifo threshold*
- `uint8_t * asrcQueue [ASRC_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSamples [ASRC_XFER_QUEUE_SIZE]`  
*Data bytes need to convert.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

#### 11.2.2.4 struct asrc\_out\_handle\_t

##### Data Fields

- `asrc_transfer_callback_t callback`  
*Callback function called at convert complete.*
- `uint32_t sampleWidth`  
*data width*
- `uint32_t fifoThreshold`  
*fifo threshold*
- `uint8_t * asrcQueue [ASRC_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSamples [ASRC_XFER_QUEUE_SIZE]`  
*Data bytes need to convert.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

#### 11.2.2.5 struct \_asrc\_handle

asrc handler

##### Data Fields

- `ASRC_Type * base`  
*base address*
- `uint32_t state`  
*Transfer status.*
- `void * userData`  
*Callback parameter passed to callback function.*
- `asrc_audio_channel_t audioDataChannels`  
*audio channel number*
- `asrc_channel_pair_t channelPair`  
*channel pair mask*
- `asrc_in_handle_t in`

- *asrc input handler*
- **asrc\_out\_handle\_t out**
- *asrc output handler*

### 11.2.3 Macro Definition Documentation

#### 11.2.3.1 #define ASRC\_XFER\_QUEUE\_SIZE (4U)

### 11.2.4 Enumeration Type Documentation

#### 11.2.4.1 anonymous enum

Enumerator

- kStatus\_ASRCIdle* ASRC is idle.
- kStatus\_ASRCInIdle* ASRC in is idle.
- kStatus\_ASRCOutIdle* ASRC out is idle.
- kStatus\_ASRCBusy* ASRC is busy.
- kStatus\_ASRCInvalidArgument* ASRC invalid argument.
- kStatus\_ASRCClockConfigureFailed* ASRC clock configure failed.
- kStatus\_ASRCChannelPairConfigureFailed* ASRC clock configure failed.
- kStatus\_ASRCConvertError* ASRC clock configure failed.
- kStatus\_ASRCNotSupport* ASRC not support.
- kStatus\_ASRCQueueFull* ASRC queue is full.
- kStatus\_ASRCOutQueueIdle* ASRC out queue is idle.
- kStatus\_ASRCInQueueIdle* ASRC in queue is idle.

#### 11.2.4.2 enum asrc\_channel\_pair\_t

Enumerator

- kASRC\_ChannelPairA* channel pair A value
- kASRC\_ChannelPairB* channel pair B value
- kASRC\_ChannelPairC* channel pair C value

#### 11.2.4.3 anonymous enum

Enumerator

- kASRC\_SampleRate\_8000HZ* asrc sample rate 8KHZ
- kASRC\_SampleRate\_11025HZ* asrc sample rate 11.025KHZ
- kASRC\_SampleRate\_12000HZ* asrc sample rate 12KHZ
- kASRC\_SampleRate\_16000HZ* asrc sample rate 16KHZ
- kASRC\_SampleRate\_22050HZ* asrc sample rate 22.05KHZ

*kASRC\_SampleRate\_24000HZ* asrc sample rate 24KHZ  
*kASRC\_SampleRate\_30000HZ* asrc sample rate 30KHZ  
*kASRC\_SampleRate\_32000HZ* asrc sample rate 32KHZ  
*kASRC\_SampleRate\_44100HZ* asrc sample rate 44.1KHZ  
*kASRC\_SampleRate\_48000HZ* asrc sample rate 48KHZ  
*kASRC\_SampleRate\_64000HZ* asrc sample rate 64KHZ  
*kASRC\_SampleRate\_88200HZ* asrc sample rate 88.2KHZ  
*kASRC\_SampleRate\_96000HZ* asrc sample rate 96KHZ  
*kASRC\_SampleRate\_128000HZ* asrc sample rate 128KHZ  
*kASRC\_SampleRate\_176400HZ* asrc sample rate 176.4KHZ  
*kASRC\_SampleRate\_192000HZ* asrc sample rate 192KHZ

#### 11.2.4.4 anonymous enum

Enumerator

*kASRC\_FPIInWaitStateInterruptEnable* FP in wait state mask.  
*kASRC\_OverLoadInterruptMask* overload interrupt mask  
*kASRC\_DataOutputCInterruptMask* data output c interrupt mask  
*kASRC\_DataOutputBInterruptMask* data output b interrupt mask  
*kASRC\_DataOutputAInterruptMask* data output a interrupt mask  
*kASRC\_DataInputCInterruptMask* data input c interrupt mask  
*kASRC\_DataInputBInterruptMask* data input b interrupt mask  
*kASRC\_DataInputAInterruptMask* data input a interrupt mask

#### 11.2.4.5 anonymous enum

Enumerator

*kASRC\_StatusDSLCounterReady* DSL counter.  
*kASRC\_StatusTaskQueueOverLoad* task queue overload  
*kASRC\_StatusPairCOutputOverLoad* pair c output overload  
*kASRC\_StatusPairBOutputOverLoad* pair b output overload  
*kASRC\_StatusPairAOutputOverLoad* pair a output overload  
*kASRC\_StatusPairCInputOverLoad* pair c input overload  
*kASRC\_StatusPairBInputOverLoad* pair b input overload  
*kASRC\_StatusPairAInputOverLoad* pair a input overload  
*kASRC\_StatusPairCOutputOverflow* pair c output overflow  
*kASRC\_StatusPairBOutputOverflow* pair b output overflow  
*kASRC\_StatusPairAOutputOverflow* pair a output overflow  
*kASRC\_StatusPairCInputUnderflow* pair c input underflow  
*kASRC\_StatusPairBInputUnderflow* pair b input under flow  
*kASRC\_StatusPairAInputUnderflow* pair a input underflow  
*kASRC\_StatusFPIInWaitState* FP in wait state.  
*kASRC\_StatusOverloadError* overload error

*kASRC\_StatusInputError* input error status  
*kASRC\_StatusOutputError* output error status  
*kASRC\_StatusPairCOutputReady* pair c output ready  
*kASRC\_StatusPairBOutputReady* pair b output ready  
*kASRC\_StatusPairAOutputReady* pair a output ready  
*kASRC\_StatusPairCInputReady* pair c input ready  
*kASRC\_StatusPairBInputReady* pair b input ready  
*kASRC\_StatusPairAInputReady* pair a input ready  
*kASRC\_StatusPairAInterrupt* pair A interrupt  
*kASRC\_StatusPairBInterrupt* pair B interrupt  
*kASRC\_StatusPairCInterrupt* pair C interrupt

#### 11.2.4.6 anonymous enum

Enumerator

*kASRC\_OutputFifoNearFull* channel pair output fifo near full  
*kASRC\_InputFifoNearEmpty* channel pair input fifo near empty

#### 11.2.4.7 enum asrc\_ratio\_t

Enumerator

*kASRC\_RatioNotUsed* ideal ratio not used  
*kASRC\_RatioUseInternalMeasured* ideal ratio use internal measure ratio, can be used for real time streaming audio  
*kASRC\_RatioUseIdealRatio* ideal ratio use manual configure ratio, can be used for the non-real time streaming audio

#### 11.2.4.8 enum asrc\_audio\_channel\_t

Enumerator

*kASRC\_ChannelsNumber1* channel number is 1  
*kASRC\_ChannelsNumber2* channel number is 2  
*kASRC\_ChannelsNumber3* channel number is 3  
*kASRC\_ChannelsNumber4* channel number is 4  
*kASRC\_ChannelsNumber5* channel number is 5  
*kASRC\_ChannelsNumber6* channel number is 6  
*kASRC\_ChannelsNumber7* channel number is 7  
*kASRC\_ChannelsNumber8* channel number is 8  
*kASRC\_ChannelsNumber9* channel number is 9  
*kASRC\_ChannelsNumber10* channel number is 10

#### 11.2.4.9 enum asrc\_data\_width\_t

Enumerator

*kASRC\_DataWidth24Bit* data width 24bit  
*kASRC\_DataWidth16Bit* data width 16bit  
*kASRC\_DataWidth8Bit* data width 8bit

#### 11.2.4.10 enum asrc\_data\_align\_t

Enumerator

*kASRC\_DataAlignMSB* data alignment MSB  
*kASRC\_DataAlignLSB* data alignment LSB

#### 11.2.4.11 enum asrc\_sign\_extension\_t

Enumerator

*kASRC\_NoSignExtension* no sign extension  
*kASRC\_SignExtension* sign extension

### 11.2.5 Function Documentation

#### 11.2.5.1 uint32\_t ASRC\_GetInstance ( ASRC\_Type \* *base* )

Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | ASRC base pointer. |
|-------------|--------------------|

#### 11.2.5.2 void ASRC\_Init ( ASRC\_Type \* *base*, uint32\_t *asrcPeripheralClock\_Hz* )

This API gates the asrc clock. The asrc module can't operate unless ASRC\_Init is called to enable the clock.

param *base* asrc base pointer. param *asrcPeripheralClock\_Hz* peripheral clock of ASRC.

#### 11.2.5.3 void ASRC\_Deinit ( ASRC\_Type \* *base* )

This API gates the ASRC clock and disable ASRC module. The ASRC module can't operate unless ASRC\_Init

Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | ASRC base pointer. |
|-------------|--------------------|

#### 11.2.5.4 void ASRC\_SoftwareReset ( ASRC\_Type \* *base* )

This software reset bit is self-clear bit, it will generate a software reset signal inside ASRC. After 9 cycles of the ASRC processing clock, this reset process will stop and this bit will cleared automatically.

Parameters

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

#### 11.2.5.5 status\_t ASRC\_SetChannelPairConfig ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair*, asrc\_channel\_pair\_config\_t \* *config*, uint32\_t *inputSampleRate*, uint32\_t *outputSampleRate* )

Parameters

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <i>base</i>             | ASRC base pointer.                                   |
| <i>channelPair</i>      | index of channel pair, reference _asrc_channel_pair. |
| <i>config</i>           | ASRC channel pair configuration pointer.             |
| <i>inputSampleRate</i>  | input audio data sample rate.                        |
| <i>outputSampleRate</i> | output audio data sample rate.                       |

#### 11.2.5.6 uint32\_t ASRC\_GetOutSamplesSize ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair*, uint32\_t *inSampleRate*, uint32\_t *outSampleRate*, uint32\_t *inSamplesize* )

Note

This API is depends on the ASRC output configuration, should be called after the ASRC\_SetChannelPairConfig.

Parameters

|                      |                           |
|----------------------|---------------------------|
| <i>base</i>          | asrc base pointer.        |
| <i>channelPair</i>   | ASRC channel pair number. |
| <i>inSampleRate</i>  | input sample rate.        |
| <i>outSampleRate</i> | output sample rate.       |
| <i>inSamplesize</i>  | input sampleS size.       |

Return values

|               |                      |
|---------------|----------------------|
| <i>output</i> | buffer size in byte. |
|---------------|----------------------|

#### 11.2.5.7 `uint32_t ASRC_MapSamplesWidth ( ASRC_Type * base, asrc_channel_pair_t channelPair, uint32_t * inWidth, uint32_t * outWidth )`

Note

This API is depends on the ASRC configuration, should be called after the ASRC\_SetChannelPair-Config.

Parameters

|                    |                           |
|--------------------|---------------------------|
| <i>base</i>        | asrc base pointer.        |
| <i>channelPair</i> | asrc channel pair index.  |
| <i>inWidth</i>     | ASRC channel pair number. |
| <i>outWidth</i>    | input sample rate.        |

Return values

|              |                    |
|--------------|--------------------|
| <i>input</i> | sample mask value. |
|--------------|--------------------|

#### 11.2.5.8 `uint32_t ASRC_GetRemainFifoSamples ( ASRC_Type * base, asrc_channel_pair_t channelPair, uint32_t * buffer, uint32_t outSampleWidth, uint32_t remainSamples )`

Parameters

|                       |                           |
|-----------------------|---------------------------|
| <i>base</i>           | asrc base pointer.        |
| <i>channelPair</i>    | ASRC channel pair number. |
| <i>buffer</i>         | input sample numbers.     |
| <i>outSampleWidth</i> | output sample width.      |
| <i>remainSamples</i>  | output sample rate.       |

Return values

|               |                 |
|---------------|-----------------|
| <i>remain</i> | samples number. |
|---------------|-----------------|

#### **11.2.5.9 static void ASRC\_ModuleEnable ( ASRC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | ASRC base pointer.               |
| <i>enable</i> | true is enable, false is disable |

#### **11.2.5.10 static void ASRC\_ChannelPairEnable ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t channelPair, bool *enable* ) [inline], [static]**

Parameters

|                    |                                                             |
|--------------------|-------------------------------------------------------------|
| <i>base</i>        | ASRC base pointer.                                          |
| <i>channelPair</i> | channel pair mask value, reference _asrc_channel_pair_mask. |
| <i>enable</i>      | true is enable, false is disable.                           |

#### **11.2.5.11 static void ASRC\_EnableInterrupt ( ASRC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| <i>base</i> | ASRC peripheral base address.                                                  |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_asrc_interrupt_mask</a> . |

#### 11.2.5.12 static void ASRC\_DisableInterrupt ( ASRC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>base</i> | ASRC peripheral base address.                                                   |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_asrc_interrupt_mask</a> . |

#### 11.2.5.13 static uint32\_t ASRC\_GetStatus ( ASRC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

ASRC Tx status flag value. Use the Status Mask to get the status value needed.

#### 11.2.5.14 static bool ASRC\_GetChannelPairInitialStatus ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channel* ) [inline], [static]

Parameters

|                |                    |
|----------------|--------------------|
| <i>base</i>    | ASRC base pointer  |
| <i>channel</i> | ASRC channel pair. |

Returns

ASRC Tx status flag value. Use the Status Mask to get the status value needed.

#### 11.2.5.15 static uint32\_t ASRC\_GetChannelPairFifoStatus ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair* ) [inline], [static]

Parameters

|                    |                    |
|--------------------|--------------------|
| <i>base</i>        | ASRC base pointer  |
| <i>channelPair</i> | ASRC channel pair. |

Returns

ASRC channel pair a fifo status flag value. Use the Status Mask to get the status value needed.

#### **11.2.5.16 static void ASRC\_ChannelPairWriteData ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair*, uint32\_t *data* ) [inline], [static]**

Note: ASRC fifo width is 24bit.

Parameters

|                    |                           |
|--------------------|---------------------------|
| <i>base</i>        | ASRC base pointer.        |
| <i>channelPair</i> | ASRC channel pair.        |
| <i>data</i>        | Data needs to be written. |

#### **11.2.5.17 static uint32\_t ASRC\_ChannelPairReadData ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair* ) [inline], [static]**

Note: ASRC fifo width is 24bit.

Parameters

|                    |                    |
|--------------------|--------------------|
| <i>base</i>        | ASRC base pointer. |
| <i>channelPair</i> | ASRC channel pair. |

Return values

|              |                 |
|--------------|-----------------|
| <i>value</i> | read from fifo. |
|--------------|-----------------|

#### **11.2.5.18 static uint32\_t ASRC\_GetInputDataRegisterAddress ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair* ) [inline], [static]**

Note: ASRC fifo width is 24bit.

Parameters

|                    |                    |
|--------------------|--------------------|
| <i>base</i>        | ASRC base pointer. |
| <i>channelPair</i> | ASRC channel pair. |

**11.2.5.19 static uint32\_t ASRC\_GetOutputDataRegisterAddress ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair* ) [inline], [static]**

Note: ASRC fifo width is 24bit.

Parameters

|                    |                    |
|--------------------|--------------------|
| <i>base</i>        | ASRC base pointer. |
| <i>channelPair</i> | ASRC channel pair. |

**11.2.5.20 status\_t ASRC\_SetIdealRatioConfig ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair*, uint32\_t *inputSampleRate*, uint32\_t *outputSampleRate* )**

The ideal ratio should be used when input clock source is not available.

Parameters

|                         |                                |
|-------------------------|--------------------------------|
| <i>base</i>             | ASRC base pointer.             |
| <i>channelPair</i>      | ASRC channel pair.             |
| <i>inputSampleRate</i>  | input audio data sample rate.  |
| <i>outputSampleRate</i> | output audio data sample rate. |

**11.2.5.21 status\_t ASRC\_TransferSetChannelPairConfig ( ASRC\_Type \* *base*, asrc\_handle\_t \* *handle*, asrc\_channel\_pair\_config\_t \* *config*, uint32\_t *inputSampleRate*, uint32\_t *outputSampleRate* )**

Parameters

|                         |                                          |
|-------------------------|------------------------------------------|
| <i>base</i>             | ASRC base pointer.                       |
| <i>handle</i>           | ASRC transactional handle pointer.       |
| <i>config</i>           | ASRC channel pair configuration pointer. |
| <i>inputSampleRate</i>  | input audio data sample rate.            |
| <i>outputSampleRate</i> | output audio data sample rate.           |

**11.2.5.22 void ASRC\_TransferCreateHandle ( ASRC\_Type \* *base*, asrc\_handle\_t \* *handle*, asrc\_channel\_pair\_t *channelPair*, asrc\_transfer\_callback\_t *inCallback*, asrc\_transfer\_callback\_t *outCallback*, void \* *userData* )**

This function initializes the handle for the ASRC transactional APIs. Call this function once to get the handle initialized.

Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | ASRC base pointer                              |
| <i>handle</i>      | ASRC handle pointer.                           |
| <i>channelPair</i> | ASRC channel pair.                             |
| <i>inCallback</i>  | Pointer to the user callback function.         |
| <i>outCallback</i> | Pointer to the user callback function.         |
| <i>userData</i>    | User parameter passed to the callback function |

**11.2.5.23 status\_t ASRC\_TransferNonBlocking ( ASRC\_Type \* *base*, asrc\_handle\_t \* *handle*, asrc\_transfer\_t \* *xfer* )**

Note

This API returns immediately after the transfer initiates, application should check the wait and check the callback status.

Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | asrc base pointer. |
|-------------|--------------------|

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>handle</i> | Pointer to the asrc_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the ASRC_transfer_t structure.                               |

Return values

|                         |                                        |
|-------------------------|----------------------------------------|
| <i>kStatus_Success</i>  | Successfully started the data receive. |
| <i>kStatus_ASRCBusy</i> | Previous receive still not finished.   |

#### 11.2.5.24 status\_t ASRC\_TransferBlocking ( ASRC\_Type \* *base*, asrc\_channel\_pair\_t *channelPair*, asrc\_transfer\_t \* *xfer* )

Note

This API returns immediately after the convert finished.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | asrc base pointer.                        |
| <i>channelPair</i> | channel pair index.                       |
| <i>xfer</i>        | Pointer to the ASRC_transfer_t structure. |

Return values

|                        |                                        |
|------------------------|----------------------------------------|
| <i>kStatus_Success</i> | Successfully started the data receive. |
|------------------------|----------------------------------------|

#### 11.2.5.25 status\_t ASRC\_TransferGetConvertedCount ( ASRC\_Type \* *base*, asrc\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | ASRC base pointer.                                                      |
| <i>handle</i> | Pointer to the asrc_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_ASRCIdle</i> | There is not a non-blocking transaction currently in progress. |

### 11.2.5.26 void ASRC\_TransferAbortConvert ( ASRC\_Type \* *base*, asrc\_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>   | ASRC base pointer.                                                      |
| <i>handle</i> | Pointer to the asrc_handle_t structure which stores the transfer state. |

### 11.2.5.27 void ASRC\_TransferTerminateConvert ( ASRC\_Type \* *base*, asrc\_handle\_t \* *handle* )

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC\_TransferAbortConvert.

Parameters

|               |                           |
|---------------|---------------------------|
| <i>base</i>   | ASRC base pointer.        |
| <i>handle</i> | ASRC eDMA handle pointer. |

### 11.2.5.28 void ASRC\_TransferHandleIRQ ( ASRC\_Type \* *base*, asrc\_handle\_t \* *handle* )

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | ASRC base pointer.                      |
| <i>handle</i> | Pointer to the asrc_handle_t structure. |

# **Chapter 12**

## **CACHE: ARMV7-M7 CACHE Memory Controller**

### **12.1 Overview**

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

The CACHE driver is created to help the user more easily operate the cache memory. The APIs for basic operations are including the following three levels:

1L. The L1 cache driver API. This level provides the level 1 caches controller drivers. The L1 caches are mainly integrated in the Core memory system, Cortex-M7 L1 caches, etc. For our Cortex-M4 series platforms, the L1 cache is the local memory controller (LMEM) which is not integrated in the Cortex-M4 processor memory system.

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

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

### **12.2 Function groups**

#### **12.2.1 L1 CACHE Operation**

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

#### **12.2.2 L2 CACHE Operation**

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

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

## Driver version

- #define **FSL\_CACHE\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 0, 4))  
*cache driver version 2.0.4.*

## Control for cortex-m7 L1 cache

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

## Unified Cache Control for all caches (cortex-m7 L1 cache + I2 pI310)

Mainly used for many drivers for easy cache operation.

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

## 12.3 Macro Definition Documentation

### 12.3.1 #define FSL\_CACHE\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

## 12.4 Function Documentation

12.4.1 **void L1CACHE\_InvalidateCacheByRange ( *uint32\_t address*, *uint32\_t size\_byte* )**

## Parameters

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

## Note

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

#### 12.4.2 static void L1CACHE\_InvalidateDCacheByRange ( *uint32\_t address*, *uint32\_t size\_byte* ) [inline], [static]

## Parameters

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

## Note

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

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

## Parameters

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

## Note

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

12.4.4 **static void L1CACHE\_CleanInvalidateDCacheByRange ( uint32\_t *address*,  
                  uint32\_t *size\_byte* ) [inline], [static]**

Parameters

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

Note

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

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

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

Parameters

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

Note

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

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

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

Parameters

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

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

## Note

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

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

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

## Parameters

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

## Note

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

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

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

## Parameters

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

## Note

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

# Chapter 13

## CACHE: CACHE Memory Controller

### 13.1 Overview

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

The CACHE driver is created to help the user more easily operate the cache memory. The APIs for basic operations are including the following three levels: 1L. The local cache driver API. This level provides the caches controller drivers.

2L. The unified cache driver API. This level provides many APIs for unified cache driver APIs for combined L1 and L2 cache maintain operations. This is provided for SDK drivers (DMA, ENET, US-DHC, etc) which should do the cache maintenance in their transactional APIs. Because in this arch, there is no L2 cache so the unified cache driver API directly calls local driver APIs.

### 13.2 Function groups

#### 13.2.1 CACHE Operation

There are Enable/Disable APIs for cache control and cache maintenance operations as Invalidate/Clean/-CleanInvalidate by all and by address range.

#### Macros

- #define `XCACHE_LINESIZE_BYTE` (FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE)  
*cache line size.*

#### Driver version

- #define `FSL_CACHE_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*cache driver version.*

#### cache control for XCACHE

- `uint32_t XCACHE_GetInstanceByAddr` (`uint32_t address`)  
*brief Returns an instance number given physical memory address.*
- `void XCACHE_EnableCache` (`XCACHE_Type *base`)  
*Enables the cache.*
- `void XCACHE_DisableCache` (`XCACHE_Type *base`)  
*Disables the cache.*
- `void XCACHE_InvalidateCache` (`XCACHE_Type *base`)  
*Invalidate the cache.*
- `void XCACHE_InvalidateCacheByRange` (`uint32_t address, uint32_t size_byte`)  
*Invalidate cache by range.*

- void **XCACHE\_CleanCache** (XCACHE\_Type \*base)  
*Cleans the cache.*
- void **XCACHE\_CleanCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans cache by range.*
- void **XCACHE\_CleanInvalidateCache** (XCACHE\_Type \*base)  
*Cleans and invalidates the cache.*
- void **XCACHE\_CleanInvalidateCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans and invalidate cache by range.*

## Unified Cache Control for all caches

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

## 13.3 Macro Definition Documentation

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

### 13.3.2 #define XCACHE\_LINESIZE\_BYTE (FSL\_FEATURE\_XCACHE\_LINESIZE\_BY- TE)

## 13.4 Function Documentation

### 13.4.1 uint32\_t XCACHE\_GetInstanceByAddr ( uint32\_t *address* )

param address The physical memory address.

Returns

XCACHE instance number starting from 0.

### 13.4.2 void XCACHE\_EnableCache ( XCACHE\_Type \* *base* )

Parameters

---

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

**13.4.3 void XCACHE\_DisableCache ( XCACHE\_Type \* *base* )**

Parameters

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

**13.4.4 void XCACHE\_InvalidateCache ( XCACHE\_Type \* *base* )**

Parameters

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

**13.4.5 void XCACHE\_InvalidateCacheByRange ( uint32\_t *address*, uint32\_t *size\_byte* )**

Parameters

|                  |                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------|
| <i>address</i>   | The physical address of cache.                                                                       |
| <i>size_byte</i> | size of the memory to be invalidated, should be larger than 0, better to align with cache line size. |

Note

Address and size should be aligned to "XCACHE\_LINESIZE\_BYTE". The startAddr here will be forced to align to XCACHE\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

**13.4.6 void XCACHE\_CleanCache ( XCACHE\_Type \* *base* )**

Parameters

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

**13.4.7 void XCACHE\_CleanCacheByRange ( uint32\_t *address*, uint32\_t *size\_byte* )**

Parameters

|                  |                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------|
| <i>address</i>   | The physical address of cache.                                                                   |
| <i>size_byte</i> | size of the memory to be cleaned, should be larger than 0, better to align with cache line size. |

Note

Address and size should be aligned to "XCACHE\_LINESIZE\_BYT". The startAddr here will be forced to align to XCACHE\_LINESIZE\_BYT if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

**13.4.8 void XCACHE\_CleanInvalidateCache ( XCACHE\_Type \* *base* )**

Parameters

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

**13.4.9 void XCACHE\_CleanInvalidateCacheByRange ( uint32\_t *address*, uint32\_t *size\_byte* )**

Parameters

|                  |                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------|
| <i>address</i>   | The physical address of cache.                                                                                   |
| <i>size_byte</i> | size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size. |

## Note

Address and size should be aligned to "XCACHE\_LINESIZE\_BYTE". The startAddr here will be forced to align to XCACHE\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

#### **13.4.10 static void ICACHE\_InvalidateByRange ( uint32\_t address, uint32\_t size\_byte ) [inline], [static]**

## Parameters

|                  |                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------|
| <i>address</i>   | The physical address.                                                                                |
| <i>size_byte</i> | size of the memory to be invalidated, should be larger than 0, better to align with cache line size. |

## Note

Address and size should be aligned to XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

#### **13.4.11 static void DCACHE\_InvalidateByRange ( uint32\_t address, uint32\_t size\_byte ) [inline], [static]**

## Parameters

|                  |                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------|
| <i>address</i>   | The physical address.                                                                                |
| <i>size_byte</i> | size of the memory to be invalidated, should be larger than 0, better to align with cache line size. |

## Note

Address and size should be aligned to XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

#### **13.4.12 static void DCACHE\_CleanByRange ( uint32\_t address, uint32\_t size\_byte ) [inline], [static]**

## Parameters

|                  |                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------|
| <i>address</i>   | The physical address.                                                                            |
| <i>size_byte</i> | size of the memory to be cleaned, should be larger than 0, better to align with cache line size. |

## Note

Address and size should be aligned to XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

### 13.4.13 static void DCACHE\_CleanInvalidateByRange ( uint32\_t *address*, uint32\_t *size\_byte* ) [inline], [static]

## Parameters

|                  |                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------|
| <i>address</i>   | The physical address.                                                                                            |
| <i>size_byte</i> | size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size. |

## Note

Address and size should be aligned to XCACHE\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_XCACHE\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

# Chapter 14

## Common Driver

### 14.1 Overview

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

#### Macros

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

#### Typedefs

- `typedef int32_t status_t`  
*Type used for all status and error return values.*

## Enumerations

- enum `_status_groups` {  
    `kStatusGroup_Generic` = 0,  
    `kStatusGroup_FLASH` = 1,  
    `kStatusGroup_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.*

## Min/max macros

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

## UINT16\_MAX(UINT32\_MAX value

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

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

## Suppress fallthrough warning macro

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

## Atomic modification

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

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

Take **SDK\_ATOMIC\_LOCAL\_CLEAR\_AND\_SET** as an example: the parameter `addr` means the address of the peripheral register or variable you want to modify atomically, the parameter `clearBits` is the bits to clear, the parameter `setBits` 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)**
- #define **SDK\_ATOMIC\_LOCAL\_SUB(addr, val)**
- #define **SDK\_ATOMIC\_LOCAL\_SET(addr, bits)**
- #define **SDK\_ATOMIC\_LOCAL\_CLEAR(addr, bits)**
- #define **SDK\_ATOMIC\_LOCAL\_TOGGLE(addr, bits)**
- #define **SDK\_ATOMIC\_LOCAL\_CLEAR\_AND\_SET(addr, clearBits, setBits)**

## Timer utilities

- #define **USEC\_TO\_COUNT(us, clockFreqInHz)** (uint64\_t)((uint64\_t)(us) \* (clockFreqInHz)) / 1000000U
 

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

*Macro to convert a raw count value to microsecond.*

- #define **MSEC\_TO\_COUNT**(ms, clockFreqInHz) (uint64\_t)((uint64\_t)(ms) \* (clockFreqInHz) / 1000U)
 

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

*Macro to convert a raw count value to millisecond.*

## Alignment variable definition macros

- #define **SDK\_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.*

## 14.2 Macro Definition Documentation

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

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

### 14.2.3 #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       |  |  |

- 14.2.4 `#define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))`
- 14.2.5 `#define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U`
- 14.2.6 `#define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U`
- 14.2.7 `#define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U`
- 14.2.8 `#define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U`
- 14.2.9 `#define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U`
- 14.2.10 `#define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U`
- 14.2.11 `#define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U`
- 14.2.12 `#define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U`
- 14.2.13 `#define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U`
- 14.2.14 `#define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U`
- 14.2.15 `#define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U`
- 14.2.16 `#define ARRAY_SIZE( x ) (sizeof(x) / sizeof((x)[0]))`
- 14.2.17 `#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

## 14.3 Typedef Documentation

- 14.3.1 `typedef int32_t status_t`

## 14.4 Enumeration Type Documentation

### 14.4.1 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\_MCU LCD* 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\_XSPI* Group number for XSPI 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.

#### 14.4.2 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.

## 14.5 Function Documentation

### 14.5.1 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. |
|------------|-------------------|

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

Parameters

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

### 14.5.3 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.      |

#### 14.5.4 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 |

#### 14.5.5 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 |

#### 14.5.6 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. |

#### 14.5.7 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. |

#### 14.5.8 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. |

#### 14.5.9 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.

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

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask.

User is required to use the [EnableGlobalIRQ\(\)](#) and [DisableGlobalIRQ\(\)](#) in pair.

### Parameters

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

# Chapter 15

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

### 15.1 Overview

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

### 15.2 Typical use case

#### 15.2.1 eDMA Operation

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

### Modules

- [eDMA core Driver](#)
- [eDMA soc Driver](#)

### Data Structures

- struct [edma\\_channel\\_Preemption\\_config\\_t](#)  
*eDMA channel priority configuration* [More...](#)
- struct [edma\\_minor\\_offset\\_config\\_t](#)  
*eDMA minor offset configuration* [More...](#)
- struct [edma\\_channel\\_config\\_t](#)  
*eDMA4 channel configuration* [More...](#)
- struct [edma\\_transfer\\_config\\_t](#)  
*edma4 channel transfer configuration* [More...](#)
- struct [edma\\_config\\_t](#)  
*eDMA global configuration structure.* [More...](#)
- struct [edma\\_handle\\_t](#)  
*eDMA transfer handle structure* [More...](#)

### Macros

- #define [FSL\\_EDMA\\_DRIVER\\_EDMA4\(1\)](#)  
*eDMA driver name*
- #define [EDMA\\_ALLOCATE\\_TCD\(name, number\)](#) AT\_NONCACHEABLE\_SECTION\_ALIGN([edma\\_tcd\\_t](#) name[number], EDMA\_TCD\_ALIGN\_SIZE)  
*Macro used for allocate edma TCD.*
- #define [DMA\\_DCHPRI\\_INDEX\(channel\)](#) (((channel) & ~0x03U) | (3U - ((channel)&0x03U)))  
*Compute the offset unit from DCHPRI3.*

## Typedefs

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

## Enumerations

- `enum {`  
 `kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),`  
 `kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }`  
*\_edma\_transfer\_status eDMA transfer status*
- `enum edma_transfer_size_t {`  
 `kEDMA_TransferSize1Bytes = 0x0U,`  
 `kEDMA_TransferSize2Bytes = 0x1U,`  
 `kEDMA_TransferSize4Bytes = 0x2U,`  
 `kEDMA_TransferSize8Bytes = 0x3U,`  
 `kEDMA_TransferSize16Bytes = 0x4U,`  
 `kEDMA_TransferSize32Bytes = 0x5U,`  
 `kEDMA_TransferSize64Bytes = 0x6U,`  
 `kEDMA_TransferSize128Bytes = 0x7U }`  
*eDMA transfer configuration*
- `enum edma_modulo_t {`

```

kEDMA_ModuloDisable = 0x0U,
kEDMA_Modulo2bytes,
kEDMA_Modulo4bytes,
kEDMA_Modulo8bytes,
kEDMA_Modulo16bytes,
kEDMA_Modulo32bytes,
kEDMA_Modulo64bytes,
kEDMA_Modulo128bytes,
kEDMA_Modulo256bytes,
kEDMA_Modulo512bytes,
kEDMA_Modulo1Kbytes,
kEDMA_Modulo2Kbytes,
kEDMA_Modulo4Kbytes,
kEDMA_Modulo8Kbytes,
kEDMA_Modulo16Kbytes,
kEDMA_Modulo32Kbytes,
kEDMA_Modulo64Kbytes,
kEDMA_Modulo128Kbytes,
kEDMA_Modulo256Kbytes,
kEDMA_Modulo512Kbytes,
kEDMA_Modulo1Mbytes,
kEDMA_Modulo2Mbytes,
kEDMA_Modulo4Mbytes,
kEDMA_Modulo8Mbytes,
kEDMA_Modulo16Mbytes,
kEDMA_Modulo32Mbytes,
kEDMA_Modulo64Mbytes,
kEDMA_Modulo128Mbytes,
kEDMA_Modulo256Mbytes,
kEDMA_Modulo512Mbytes,
kEDMA_Modulo1Gbytes,
kEDMA_Modulo2Gbytes }

eDMA modulo configuration
• enum edma_bandwidth_t {
 kEDMA_BandwidthStallNone = 0x0U,
 kEDMA_BandwidthStall4Cycle = 0x2U,
 kEDMA_BandwidthStall8Cycle = 0x3U }

Bandwidth control.
• enum edma_channel_link_type_t {
 kEDMA_LinkNone = 0x0U,
 kEDMA_MinorLink,
 kEDMA_MajorLink }

Channel link type.
• enum {

```

```

kEDMA_DoneFlag = 0x1U,
kEDMA_ErrorFlag = 0x2U,
kEDMA_InterruptFlag = 0x4U }
 _edma_channel_status_flags eDMA channel status flags.
• enum {
 kEDMA_DestinationBusErrorFlag = DMA_ERR_DBE_FLAG,
 kEDMA_SourceBusErrorFlag = DMA_ERR_SBE_FLAG,
 kEDMA_ScatterGatherErrorFlag = DMA_ERR_SGE_FLAG,
 kEDMA_NbytesErrorFlag = DMA_ERR_NCE_FLAG,
 kEDMA_DestinationOffsetErrorFlag = DMA_ERR_DOE_FLAG,
 kEDMA_DestinationAddressErrorFlag = DMA_ERR_DAE_FLAG,
 kEDMA_SourceOffsetErrorFlag = DMA_ERR_SOE_FLAG,
 kEDMA_SourceAddressErrorFlag = DMA_ERR_SAE_FLAG,
 kEDMA_ErrorChannelFlag = DMA_ERR_ERRCHAN_FLAG,
 kEDMA_TransferCanceledFlag = DMA_ERR_ECX_FLAG,
 kEDMA_ValidFlag = (int)DMA_ERR_FLAG }
 _edma_error_status_flags eDMA channel error status flags.
• enum {
 kEDMA_ErrorInterruptEnable = 0x1U,
 kEDMA_MajorInterruptEnable = DMA_CSR_INTMAJOR_MASK,
 kEDMA_HalfInterruptEnable = DMA_CSR_INTHALF_MASK }
 _edma_interrupt_enable eDMA interrupt source
• enum edma_transfer_type_t {
 kEDMA_MemoryToMemory = 0x0U,
 kEDMA_PeripheralToMemory,
 kEDMA_MemoryToPeripheral,
 kEDMA_PeripheralToPeripheral }

 eDMA transfer type
• enum edma_channel_memory_attribute_t {
 kEDMA_ChannelNoWriteNoReadNoCacheNoBuffer,
 kEDMA_ChannelNoWriteNoReadNoCacheBufferable,
 kEDMA_ChannelNoWriteNoReadCacheableNoBuffer,
 kEDMA_ChannelNoWriteNoReadCacheableBufferable,
 kEDMA_ChannelNoWriteReadNoCacheNoBuffer,
 kEDMA_ChannelNoWriteReadNoCacheBufferable,
 kEDMA_ChannelNoWriteReadCacheableNoBuffer,
 kEDMA_ChannelNoWriteReadCacheableBufferable,
 kEDMA_ChannelWriteNoReadNoCacheNoBuffer,
 kEDMA_ChannelWriteNoReadNoCacheBufferable,
 kEDMA_ChannelWriteNoReadCacheableNoBuffer,
 kEDMA_ChannelWriteNoReadCacheableBufferable,
 kEDMA_ChannelWriteReadNoCacheNoBuffer,
 kEDMA_ChannelWriteReadNoCacheBufferable,
 kEDMA_ChannelWriteReadCacheableNoBuffer,
 kEDMA_ChannelWriteReadCacheableBufferable }

 eDMA channel memory attribute

```

- enum `edma_channel_swap_size_t` {
   
  `kEDMA_ChannelSwapDisabled` = 0x0U,
   
  `kEDMA_ChannelReadWith8bitSwap` = 0x1U,
   
  `kEDMA_ChannelReadWith16bitSwap` = 0x2U,
   
  `kEDMA_ChannelReadWith32bitSwap` = 0x3U,
   
  `kEDMA_ChannelWriteWith8bitSwap` = 0x9U,
   
  `kEDMA_ChannelWriteWith16bitSwap` = 0x10U,
   
  `kEDMA_ChannelWriteWith32bitSwap` = 0x11U }
- eDMA4 channel swap size*
- enum {
   
  `kEDMA_AttributeOutput` = DMA\_CH\_SBR\_ATTR\_MASK,
   
  `kEDMA_PrivilegedAccessLevel` = DMA\_CH\_SBR\_PAL\_MASK,
   
  `kEDMA_MasterId` }
- eDMA channel system bus information, \_edma\_channel\_sys\_bus\_info*
- enum `edma_channel_access_type_t` {
   
  `kEDMA_ChannelDataAccess` = 0x0U,
   
  `kEDMA_ChannelInstructionAccess` = 0x1U }
- eDMA4 channel access type*
- enum `edma_channel_protection_level_t` {
   
  `kEDMA_ChannelProtectionLevelUser` = 0x0U,
   
  `kEDMA_ChannelProtectionLevelPrivileged` = 0x1U }
- eDMA4 channel protection level*
- enum `edma_channel_security_level_t` {
   
  `kEDMA_ChannelSecurityLevelNonSecure` = 0x0U,
   
  `kEDMA_ChannelSecurityLevelSecure` = 0x1U }
- eDMA4 channel security level*

## Driver version

- #define `FSL_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 9, 0))
- eDMA driver version*

## eDMA initialization and de-initialization

- void `EDMA_Init` (EDMA\_Type \*base, const `edma_config_t` \*config)
   
    *Initializes the eDMA peripheral.*
- void `EDMA_Deinit` (EDMA\_Type \*base)
   
    *Deinitializes the eDMA peripheral.*
- void `EDMA_InstallTCD` (EDMA\_Type \*base, uint32\_t channel, const `edma_tcd_t` \*tcd)
   
    *Push content of TCD structure into hardware TCD register.*
- void `EDMA_GetDefaultConfig` (`edma_config_t` \*config)
   
    *Gets the eDMA default configuration structure.*

## eDMA Channel Operation

- void `EDMA_InitChannel` (EDMA\_Type \*base, uint32\_t channel, `edma_channel_config_t` \*channelConfig)
   
    *EDMA Channel initialization.*

- static void `EDMA_SetChannelMemoryAttribute` (EDMA\_Type \*base, uint32\_t channel, `edma_channel_memory_attribute_t` writeAttribute, `edma_channel_memory_attribute_t` readAttribute)  
*Set channel memory attribute.*
- static void `EDMA_SetChannelSignExtension` (EDMA\_Type \*base, uint32\_t channel, uint8\_t position)  
*Set channel sign extension.*
- static void `EDMA_SetChannelSwapSize` (EDMA\_Type \*base, uint32\_t channel, `edma_channel_swap_size_t` swapSize)  
*Set channel swap size.*
- static void `EDMA_SetChannelAccessType` (EDMA\_Type \*base, uint32\_t channel, `edma_channel_access_type_t` channelAccessType)  
*Set channel access type.*
- static void `EDMA_SetChannelMux` (EDMA\_Type \*base, uint32\_t channel, int32\_t channelRequestSource)  
*Set channel request source.*
- static uint32\_t `EDMA_GetChannelSystemBusInformation` (EDMA\_Type \*base, uint32\_t channel)  
*Gets the channel identification and attribute information on the system bus interface.*
- static void `EDMA_EnableChannelMasterIDReplication` (EDMA\_Type \*base, uint32\_t channel, bool enable)  
*Set channel master ID replication.*
- static void `EDMA_SetChannelSecurityLevel` (EDMA\_Type \*base, uint32\_t channel, `edma_channel_security_level_t` level)  
*Set channel security level.*
- static void `EDMA_SetChannelProtectionLevel` (EDMA\_Type \*base, uint32\_t channel, `edma_channel_protection_level_t` level)  
*Set channel security level.*
- void `EDMA_ResetChannel` (EDMA\_Type \*base, uint32\_t channel)  
*Sets all TCD registers to default values.*
- void `EDMA_SetTransferConfig` (EDMA\_Type \*base, uint32\_t channel, const `edma_transfer_config_t` \*config, `edma_tcd_t` \*nextTcd)  
*Configures the eDMA transfer attribute.*
- void `EDMA_SetMinorOffsetConfig` (EDMA\_Type \*base, uint32\_t channel, const `edma_minor_offset_config_t` \*config)  
*Configures the eDMA minor offset feature.*
- void `EDMA_SetChannelPreemptionConfig` (EDMA\_Type \*base, uint32\_t channel, const `edma_channel_Preemption_config_t` \*config)  
*Configures the eDMA channel preemption feature.*
- void `EDMA_SetChannelLink` (EDMA\_Type \*base, uint32\_t channel, `edma_channel_link_type_t` type, uint32\_t linkedChannel)  
*Sets the channel link for the eDMA transfer.*
- void `EDMA_SetBandWidth` (EDMA\_Type \*base, uint32\_t channel, `edma_bandwidth_t` bandwidth)  
*Sets the bandwidth for the eDMA transfer.*
- void `EDMA_SetModulo` (EDMA\_Type \*base, uint32\_t channel, `edma_modulo_t` srcModulo, `edma_modulo_t` destModulo)  
*Sets the source modulo and the destination modulo for the eDMA transfer.*
- static void `EDMA_EnableAutoStopRequest` (EDMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an auto stop request for the eDMA transfer.*
- void `EDMA_EnableChannelInterrupts` (EDMA\_Type \*base, uint32\_t channel, uint32\_t mask)

*Enables the interrupt source for the eDMA transfer.*

- void **EDMA\_DisableChannelInterrupts** (EDMA\_Type \*base, uint32\_t channel, uint32\_t mask)

*Disables the interrupt source for the eDMA transfer.*

- void **EDMA\_SetMajorOffsetConfig** (EDMA\_Type \*base, uint32\_t channel, int32\_t sourceOffset, int32\_t destOffset)

*Configures the eDMA channel TCD major offset feature.*

## eDMA TCD Operation

- void **EDMA\_ConfigChannelSoftwareTCD** (edma\_tcd\_t \*tcd, const edma\_transfer\_config\_t \*transfer)

*Sets TCD fields according to the user's channel transfer configuration structure, `edma_transfer_config_t`.*

- void **EDMA\_TcdReset** (edma\_tcd\_t \*tcd)

*Sets all fields to default values for the TCD structure.*

- void **EDMA\_TcdSetTransferConfig** (edma\_tcd\_t \*tcd, const edma\_transfer\_config\_t \*config, edma\_tcd\_t \*nextTcd)

*Configures the eDMA TCD transfer attribute.*

- void **EDMA\_TcdSetMinorOffsetConfig** (edma\_tcd\_t \*tcd, const edma\_minor\_offset\_config\_t \*config)

*Configures the eDMA TCD minor offset feature.*

- void **EDMA\_TcdSetChannelLink** (edma\_tcd\_t \*tcd, edma\_channel\_link\_type\_t type, uint32\_t linkedChannel)

*Sets the channel link for the eDMA TCD.*

- static void **EDMA\_TcdSetBandWidth** (edma\_tcd\_t \*tcd, edma\_bandwidth\_t bandwidth)

*Sets the bandwidth for the eDMA TCD.*

- void **EDMA\_TcdSetModulo** (edma\_tcd\_t \*tcd, edma\_modulo\_t srcModulo, edma\_modulo\_t destModulo)

*Sets the source modulo and the destination modulo for the eDMA TCD.*

- static void **EDMA\_TcdEnableAutoStopRequest** (edma\_tcd\_t \*tcd, bool enable)

*Sets the auto stop request for the eDMA TCD.*

- void **EDMA\_TcdEnableInterrupts** (edma\_tcd\_t \*tcd, uint32\_t mask)

*Enables the interrupt source for the eDMA TCD.*

- void **EDMA\_TcdDisableInterrupts** (edma\_tcd\_t \*tcd, uint32\_t mask)

*Disables the interrupt source for the eDMA TCD.*

- void **EDMA\_TcdSetMajorOffsetConfig** (edma\_tcd\_t \*tcd, int32\_t sourceOffset, int32\_t destOffset)

*Configures the eDMA TCD major offset feature.*

## eDMA Channel Transfer Operation

- static void **EDMA\_EnableChannelRequest** (EDMA\_Type \*base, uint32\_t channel)

*Enables the eDMA hardware channel request.*

- static void **EDMA\_DisableChannelRequest** (EDMA\_Type \*base, uint32\_t channel)

*Disables the eDMA hardware channel request.*

- static void **EDMA\_TriggerChannelStart** (EDMA\_Type \*base, uint32\_t channel)

*Starts the eDMA transfer by using the software trigger.*

## eDMA Channel Status Operation

- uint32\_t **EDMA\_GetRemainingMajorLoopCount** (EDMA\_Type \*base, uint32\_t channel)

*Gets the remaining major loop count from the eDMA current channel TCD.*

- static uint32\_t **EDMA\_GetErrorStatusFlags** (EDMA\_Type \*base)  
*Gets the eDMA channel error status flags.*
- uint32\_t **EDMA\_GetChannelStatusFlags** (EDMA\_Type \*base, uint32\_t channel)  
*Gets the eDMA channel status flags.*
- void **EDMA\_ClearChannelStatusFlags** (EDMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Clears the eDMA channel status flags.*

## eDMA Transactional Operation

- void **EDMA\_CreateHandle** (edma\_handle\_t \*handle, EDMA\_Type \*base, uint32\_t channel)  
*Creates the eDMA handle.*
- void **EDMA\_InstallTCDMemory** (edma\_handle\_t \*handle, edma\_tcd\_t \*tcdPool, uint32\_t tcdSize)  
*Installs the TCDs memory pool into the eDMA handle.*
- void **EDMA\_SetCallback** (edma\_handle\_t \*handle, edma\_callback callback, void \*userData)  
*Installs a callback function for the eDMA transfer.*
- void **EDMA\_PreparesTransferConfig** (edma\_transfer\_config\_t \*config, void \*srcAddr, uint32\_t srcWidth, int16\_t srcOffset, void \*destAddr, uint32\_t destWidth, int16\_t destOffset, uint32\_t bytesEachRequest, uint32\_t transferBytes)  
*Prepares the eDMA transfer structure configurations.*
- void **EDMA\_PreparesTransfer** (edma\_transfer\_config\_t \*config, void \*srcAddr, uint32\_t srcWidth, void \*destAddr, uint32\_t destWidth, uint32\_t bytesEachRequest, uint32\_t transferBytes, edma\_transfer\_type\_t type)  
*Prepares the eDMA transfer structure.*
- void **EDMA\_PreparesTransferTCD** (edma\_handle\_t \*handle, edma\_tcd\_t \*tcd, void \*srcAddr, uint32\_t srcWidth, int16\_t srcOffset, void \*destAddr, uint32\_t destWidth, int16\_t destOffset, uint32\_t bytesEachRequest, uint32\_t transferBytes, edma\_tcd\_t \*nextTcd)  
*Prepares the eDMA transfer content descriptor.*
- status\_t **EDMA\_SubmitTransferTCD** (edma\_handle\_t \*handle, const edma\_tcd\_t \*tcd)  
*Submits the eDMA transfer content descriptor.*
- status\_t **EDMA\_SubmitTransfer** (edma\_handle\_t \*handle, const edma\_transfer\_config\_t \*config)  
*Submits the eDMA transfer request.*
- status\_t **EDMA\_SubmitLoopTransfer** (edma\_handle\_t \*handle, edma\_transfer\_config\_t \*transfer, uint32\_t transferLoopCount)  
*Submits the eDMA scatter gather transfer configurations.*
- void **EDMA\_StartTransfer** (edma\_handle\_t \*handle)  
*eDMA starts transfer.*
- void **EDMA\_StopTransfer** (edma\_handle\_t \*handle)  
*eDMA stops transfer.*
- void **EDMA\_AbortTransfer** (edma\_handle\_t \*handle)  
*eDMA aborts transfer.*
- static uint32\_t **EDMA\_GetUnusedTCDNumber** (edma\_handle\_t \*handle)  
*Get unused TCD slot number.*
- static uint32\_t **EDMA\_GetNextTCDAddress** (edma\_handle\_t \*handle)  
*Get the next tcd address.*
- void **EDMA\_HandleIRQ** (edma\_handle\_t \*handle)  
*eDMA IRQ handler for the current major loop transfer completion.*

## 15.3 Data Structure Documentation

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

#### Data Fields

- bool `enableChannelPreemption`  
*If true: a channel can be suspended by other channel with higher priority.*
- bool `enablePreemptAbility`  
*If true: a channel can suspend other channel with low priority.*
- uint8\_t `channelPriority`  
*Channel priority.*

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

#### Data Fields

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

#### Field Documentation

- (1) `bool edma_minor_offset_config_t::enableSrcMinorOffset`
- (2) `bool edma_minor_offset_config_t::enableDestMinorOffset`
- (3) `uint32_t edma_minor_offset_config_t::minorOffset`

### 15.3.3 struct edma\_channel\_config\_t

#### Data Fields

- `edma_channel_Preemption_config_t channelPreemptionConfig`  
*channel preemption configuration*
- `edma_channel_memory_attribute_t channelReadMemoryAttribute`  
*channel memory read attribute configuration*
- `edma_channel_memory_attribute_t channelWriteMemoryAttribute`  
*channel memory write attribute configuration*
- `edma_channel_swap_size_t channelSwapSize`  
*channel swap size configuration*
- `edma_channel_access_type_t channelAccessType`  
*channel access type configuration*
- `uint8_t channelDataSignExtensionBitPosition`  
*channel data sign extension bit position configuration*

- int **channelRequestSource**  
*hardware service request source for the channel*
- bool **enableMasterIDReplication**  
*enable master ID replication*
- **edma\_channel\_security\_level\_t securityLevel**  
*security level*
- **edma\_channel\_protection\_level\_t protectionLevel**  
*protection level*

#### 15.3.4 struct edma\_transfer\_config\_t

The transfer configuration structure support full feature configuration of the transfer control descriptor.

##### Note

User should pay attention to the transfer size alignment limitation

1. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0
2. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligned with transfer width
3. the totalBytes should align with the bytesEachRequest
4. the srcAddr should align with the srcWidthOfEachTransfer
5. the dstAddr should align with the dstWidthOfEachTransfer
6. the srcAddr should align with srcAddrModulo if modulo feature is enabled
7. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the edma4 interfaces will generate assert error.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enableChannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

#### Data Fields

- uint32\_t **srcAddr**  
*Source data address.*
- uint32\_t **destAddr**  
*Destination data address.*

- **edma\_transfer\_size\_t srcTransferSize**  
*Source data transfer size.*
- **edma\_transfer\_size\_t destTransferSize**  
*Destination data transfer size.*
- **int16\_t srcOffset**  
*Sign-extended offset value in byte unit applied to the current source address to form the next-state value as each source read is completed.*
- **int16\_t destOffset**  
*Sign-extended offset value in byte unit applied to the current destination address to form the next-state value as each destination write is completed.*
- **uint32\_t minorLoopBytes**  
*bytes in each minor loop or each request range: 1 - (2^30 - 1) when minor loop mapping is enabled range: 1 - (2^10 - 1) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - (2^32 - 1) when minor loop mapping is disabled*
- **uint32\_t majorLoopCounts**  
*minor loop counts in each major loop, should be 1 at least for each transfer range: (0 - (2^15 - 1)) when minor loop channel link is disabled range: (0 - (2^9 - 1)) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajorLoop \* bytesEachMinorLoop*
- **uint16\_t enabledInterruptMask**  
*channel interrupt to enable, can be OR'ed value of \_edma\_interrupt\_enable*
- **edma\_modulo\_t srcAddrModulo**  
*source circular data queue range*
- **int32\_t srcMajorLoopOffset**  
*source major loop offset*
- **edma\_modulo\_t dstAddrModulo**  
*destination circular data queue range*
- **int32\_t dstMajorLoopOffset**  
*destination major loop offset*
- **bool enableSrcMinorLoopOffset**  
*enable source minor loop offset*
- **bool enableDstMinorLoopOffset**  
*enable dest minor loop offset*
- **int32\_t minorLoopOffset**  
*burst offset, the offset will be applied after minor loop update*
- **bool enableChannelMajorLoopLink**  
*channel link when major loop complete*
- **uint32\_t majorLoopLinkChannel**  
*major loop link channel number*
- **bool enableChannelMinorLoopLink**  
*channel link when minor loop complete*
- **uint32\_t minorLoopLinkChannel**  
*minor loop link channel number*
- **edma\_tcd\_t \* linkTCD**  
*pointer to the link transfer control descriptor*

**Field Documentation**

- (1) **uint32\_t edma\_transfer\_config\_t::srcAddr**
- (2) **uint32\_t edma\_transfer\_config\_t::destAddr**
- (3) **edma\_transfer\_size\_t edma\_transfer\_config\_t::srcTransferSize**
- (4) **edma\_transfer\_size\_t edma\_transfer\_config\_t::destTransferSize**
- (5) **int16\_t edma\_transfer\_config\_t::destOffset**

**15.3.5 struct edma\_config\_t****Data Fields**

- **bool enableMasterIdReplication**  
*Enable (true) master ID replication.*
- **bool enableGlobalChannelLink**  
*Enable(true) channel linking is available and controlled by each channel's link settings.*
- **bool enableHaltOnError**  
*Enable (true) transfer halt on error.*
- **bool enableDebugMode**  
*Enable(true) eDMA4 debug mode.*
- **bool enableRoundRobinArbitration**  
*Enable(true) channel linking is available and controlled by each channel's link settings.*
- **edma\_channel\_config\_t \* channelConfig [FSL\_FEATURE\_EDMA\_MODULE\_CHANNEL]**  
*channel preemption configuration*

**Field Documentation**

- (1) **bool edma\_config\_t::enableMasterIdReplication**

If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

- (2) **bool edma\_config\_t::enableGlobalChannelLink**

- (3) **bool edma\_config\_t::enableHaltOnError**

Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

- (4) **bool edma\_config\_t::enableDebugMode**

When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

(5) `bool edma_config_t::enableRoundRobinArbitration`

### 15.3.6 struct edma\_handle\_t

#### Data Fields

- `edma_callback callback`  
*Callback function for major count exhausted.*
- `void * userData`  
*Callback function parameter.*
- `EDMA_ChannelType * channelBase`  
*eDMA peripheral channel base address.*
- `EDMA_Type * base`  
*eDMA peripheral base address*
- `EDMA_TCDType * tcdBase`  
*eDMA peripheral tcd base address.*
- `edma_tcd_t * tcdPool`  
*Pointer to memory stored TCDs.*
- `uint32_t channel`  
*eDMA channel number.*
- `volatile int8_t header`  
*The first TCD index.*
- `volatile int8_t tail`  
*The last TCD index.*
- `volatile int8_t tcdUsed`  
*The number of used TCD slots.*
- `volatile int8_t tcdSize`  
*The total number of TCD slots in the queue.*

#### Field Documentation

(1) `edma_callback edma_handle_t::callback`

(2) `void* edma_handle_t::userData`

(3) `EDMA_ChannelType* edma_handle_t::channelBase`

(4) `EDMA_TCDType* edma_handle_t::tcdBase`

(5) `edma_tcd_t* edma_handle_t::tcdPool`

(6) `uint32_t edma_handle_t::channel`

(7) `volatile int8_t edma_handle_t::header`

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

(8) `volatile int8_t edma_handle_t::tail`

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

**(9) volatile int8\_t edma\_handle\_t::tcdUsed**

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

**(10) volatile int8\_t edma\_handle\_t::tcdSize****15.4 Macro Definition Documentation****15.4.1 #define FSL\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 9, 0))**

Version 2.9.1.

**15.5 Typedef Documentation****15.5.1 typedef edma\_core\_tcd\_t edma\_tcd\_t**

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

**15.5.2 typedef void(\* edma\_callback)(struct \_edma\_handle \*handle, void \*userData, bool transferDone, uint32\_t tclds)**

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

Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>       | EDMA handle pointer, users shall not touch the values inside.                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>userData</i>     | The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.                                                                                                                                                                                                                                                                                                              |
| <i>transferDone</i> | If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDM-A register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers. |

|             |                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcds</i> | How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this. |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 15.6 Enumeration Type Documentation

### 15.6.1 anonymous enum

Enumerator

*kStatus\_EDMA\_QueueFull* TCD queue is full.

*kStatus\_EDMA\_Busy* Channel is busy and can't handle the transfer request.

### 15.6.2 enum edma\_transfer\_size\_t

Enumerator

*kEDMA\_TransferSize1Bytes* Source/Destination data transfer size is 1 byte every time.

*kEDMA\_TransferSize2Bytes* Source/Destination data transfer size is 2 bytes every time.

*kEDMA\_TransferSize4Bytes* Source/Destination data transfer size is 4 bytes every time.

*kEDMA\_TransferSize8Bytes* Source/Destination data transfer size is 8 bytes every time.

*kEDMA\_TransferSize16Bytes* Source/Destination data transfer size is 16 bytes every time.

*kEDMA\_TransferSize32Bytes* Source/Destination data transfer size is 32 bytes every time.

*kEDMA\_TransferSize64Bytes* Source/Destination data transfer size is 64 bytes every time.

*kEDMA\_TransferSize128Bytes* Source/Destination data transfer size is 128 bytes every time.

### 15.6.3 enum edma\_modulo\_t

Enumerator

*kEDMA\_ModuloDisable* Disable modulo.

*kEDMA\_Modulo2bytes* Circular buffer size is 2 bytes.

*kEDMA\_Modulo4bytes* Circular buffer size is 4 bytes.

*kEDMA\_Modulo8bytes* Circular buffer size is 8 bytes.

*kEDMA\_Modulo16bytes* Circular buffer size is 16 bytes.

*kEDMA\_Modulo32bytes* Circular buffer size is 32 bytes.

*kEDMA\_Modulo64bytes* Circular buffer size is 64 bytes.

*kEDMA\_Modulo128bytes* Circular buffer size is 128 bytes.

*kEDMA\_Modulo256bytes* Circular buffer size is 256 bytes.

*kEDMA\_Modulo512bytes* Circular buffer size is 512 bytes.

*kEDMA\_Modulo1Kbytes* Circular buffer size is 1 K bytes.

*kEDMA\_Modulo2Kbytes* Circular buffer size is 2 K bytes.

*kEDMA\_Modulo4Kbytes* Circular buffer size is 4 K bytes.

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

#### 15.6.4 enum edma\_bandwidth\_t

Enumerator

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

#### 15.6.5 enum edma\_channel\_link\_type\_t

Enumerator

*kEDMA\_LinkNone* No channel link.  
*kEDMA\_MinorLink* Channel link after each minor loop.  
*kEDMA\_MajorLink* Channel link while major loop count exhausted.

#### 15.6.6 anonymous enum

Enumerator

*kEDMA\_DoneFlag* DONE flag, set while transfer finished, CITER value exhausted.  
*kEDMA\_ErrorFlag* eDMA error flag, an error occurred in a transfer  
*kEDMA\_InterruptFlag* eDMA interrupt flag, set while an interrupt occurred of this channel

### 15.6.7 anonymous enum

Enumerator

- kEDMA\_DestinationBusErrorFlag*** Bus error on destination address.
- kEDMA\_SourceBusErrorFlag*** Bus error on the source address.
- kEDMA\_ScatterGatherErrorFlag*** Error on the Scatter/Gather address, not 32byte aligned.
- kEDMA\_NbytesErrorFlag*** NBYTES/CITER configuration error.
- kEDMA\_DestinationOffsetErrorFlag*** Destination offset not aligned with destination size.
- kEDMA\_DestinationAddressErrorFlag*** Destination address not aligned with destination size.
- kEDMA\_SourceOffsetErrorFlag*** Source offset not aligned with source size.
- kEDMA\_SourceAddressErrorFlag*** Source address not aligned with source size.
- kEDMA\_ErrorChannelFlag*** Error channel number of the cancelled channel number.
- kEDMA\_TransferCanceledFlag*** Transfer cancelled.
- kEDMA\_ValidFlag*** No error occurred, this bit is 0. Otherwise, it is 1.

### 15.6.8 anonymous enum

Enumerator

- kEDMA\_ErrorInterruptEnable*** Enable interrupt while channel error occurs.
- kEDMA\_MajorInterruptEnable*** Enable interrupt while major count exhausted.
- kEDMA\_HalfInterruptEnable*** Enable interrupt while major count to half value.

### 15.6.9 enum edma\_transfer\_type\_t

Enumerator

- kEDMA\_MemoryToMemory*** Transfer from memory to memory.
- kEDMA\_PeripheralToMemory*** Transfer from peripheral to memory.
- kEDMA\_MemoryToPeripheral*** Transfer from memory to peripheral.
- kEDMA\_PeripheralToPeripheral*** Transfer from Peripheral to peripheral.

### 15.6.10 enum edma\_channel\_memory\_attribute\_t

Enumerator

- kEDMA\_ChannelNoWriteNoReadNoCacheNoBuffer*** No write allocate, no read allocate, non-cacheable, non-bufferable.
- kEDMA\_ChannelNoWriteNoReadNoCacheBufferable*** No write allocate, no read allocate, non-cacheable, bufferable.
- kEDMA\_ChannelNoWriteNoReadCacheableNoBuffer*** No write allocate, no read allocate, cacheable, non-bufferable.

***kEDMA\_ChannelNoWriteNoReadCacheableBufferable*** No write allocate, no read allocate, cacheable, bufferable.

***kEDMA\_ChannelNoWriteReadNoCacheNoBuffer*** No write allocate, read allocate, non-cacheable, non-bufferable.

***kEDMA\_ChannelNoWriteReadNoCacheBufferable*** No write allocate, read allocate, non-cacheable, bufferable.

***kEDMA\_ChannelNoWriteReadCacheableNoBuffer*** No write allocate, read allocate, cacheable, non-bufferable.

***kEDMA\_ChannelNoWriteReadCacheableBufferable*** No write allocate, read allocate, cacheable, bufferable.

***kEDMA\_ChannelWriteNoReadNoCacheNoBuffer*** write allocate, no read allocate, non-cacheable, non-bufferable.

***kEDMA\_ChannelWriteNoReadNoCacheBufferable*** write allocate, no read allocate, non-cacheable, bufferable.

***kEDMA\_ChannelWriteNoReadCacheableNoBuffer*** write allocate, no read allocate, cacheable, non-bufferable.

***kEDMA\_ChannelWriteNoReadCacheableBufferable*** write allocate, no read allocate, cacheable, bufferable.

***kEDMA\_ChannelWriteReadNoCacheNoBuffer*** write allocate, read allocate, non-cacheable, non-bufferable.

***kEDMA\_ChannelWriteReadNoCacheBufferable*** write allocate, read allocate, non-cacheable, bufferable.

***kEDMA\_ChannelWriteReadCacheableNoBuffer*** write allocate, read allocate, cacheable, non-bufferable.

***kEDMA\_ChannelWriteReadCacheableBufferable*** write allocate, read allocate, cacheable, bufferable.

### 15.6.11 enum edma\_channel\_swap\_size\_t

Enumerator

***kEDMA\_ChannelSwapDisabled*** Swap is disabled.

***kEDMA\_ChannelReadWith8bitSwap*** Swap occurs with respect to the read 8bit.

***kEDMA\_ChannelReadWith16bitSwap*** Swap occurs with respect to the read 16bit.

***kEDMA\_ChannelReadWith32bitSwap*** Swap occurs with respect to the read 32bit.

***kEDMA\_ChannelWriteWith8bitSwap*** Swap occurs with respect to the write 8bit.

***kEDMA\_ChannelWriteWith16bitSwap*** Swap occurs with respect to the write 16bit.

***kEDMA\_ChannelWriteWith32bitSwap*** Swap occurs with respect to the write 32bit.

### 15.6.12 anonymous enum

Enumerator

***kEDMA\_AttributeOutput*** DMA's AHB system bus attribute output value.

***kEDMA\_PrivilegedAccessLevel*** Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

***kEDMA\_MasterId*** DMA's master ID when channel is active and master ID replication is enabled.

### 15.6.13 enum edma\_channel\_access\_type\_t

Enumerator

***kEDMA\_ChannelDataAccess*** Data access for eDMA4 transfers.

***kEDMA\_ChannelInstructionAccess*** Instruction access for eDMA4 transfers.

### 15.6.14 enum edma\_channel\_protection\_level\_t

Enumerator

***kEDMA\_ChannelProtectionLevelUser*** user protection level for eDMA transfers.

***kEDMA\_ChannelProtectionLevelPrivileged*** Privileged protection level eDMA transfers.

### 15.6.15 enum edma\_channel\_security\_level\_t

Enumerator

***kEDMA\_ChannelSecurityLevelNonSecure*** non secure level for eDMA transfers.

***kEDMA\_ChannelSecurityLevelSecure*** secure level for eDMA transfers.

## 15.7 Function Documentation

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

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

Parameters

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

Note

This function enables the minor loop map feature.

### **15.7.2 void EDMA\_Deinit( EDMA\_Type \* *base* )**

This function gates the eDMA clock.

Parameters

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

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

Parameters

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

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

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

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

Parameters

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

### 15.7.5 void EDMA\_InitChannel ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **edma\_channel\_config\_t** \* *channelConfig* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | eDMA4 peripheral base address. |
|-------------|--------------------------------|

|                      |                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------|
| <i>channel</i>       | eDMA4 channel number.                                                                                   |
| <i>channelConfig</i> | pointer to user's eDMA4 channel config structure, see <a href="#">edma_channel_config_t</a> for detail. |

**15.7.6 static void EDMA\_SetChannelMemoryAttribute ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **edma\_channel\_memory\_attribute\_t** *writeAttribute*, **edma\_channel\_memory\_attribute\_t** *readAttribute* ) [inline], [static]**

Parameters

|                       |                                                 |
|-----------------------|-------------------------------------------------|
| <i>base</i>           | eDMA4 peripheral base address.                  |
| <i>channel</i>        | eDMA4 channel number.                           |
| <i>writeAttribute</i> | Attributes associated with a write transaction. |
| <i>readAttribute</i>  | Attributes associated with a read transaction.  |

**15.7.7 static void EDMA\_SetChannelSignExtension ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **uint8\_t** *position* ) [inline], [static]**

Parameters

|                 |                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| <i>base</i>     | eDMA4 peripheral base address.                                                              |
| <i>channel</i>  | eDMA4 channel number.                                                                       |
| <i>position</i> | A non-zero value specifying the sign extend bit position. If 0, sign extension is disabled. |

**15.7.8 static void EDMA\_SetChannelSwapSize ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **edma\_channel\_swap\_size\_t** *swapSize* ) [inline], [static]**

Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>base</i>     | eDMA4 peripheral base address.                                                   |
| <i>channel</i>  | eDMA4 channel number.                                                            |
| <i>swapSize</i> | Swap occurs with respect to the specified transfer size. If 0, swap is disabled. |

15.7.9 **static void EDMA\_SetChannelAccessType ( EDMA\_Type \* *base*, uint32\_t *channel*, edma\_channel\_access\_type\_t *channelAccessType* ) [inline], [static]**

Parameters

|                           |                                                                         |
|---------------------------|-------------------------------------------------------------------------|
| <i>base</i>               | eDMA4 peripheral base address.                                          |
| <i>channel</i>            | eDMA4 channel number.                                                   |
| <i>channelAccess-Type</i> | eDMA4's transactions type on the system bus when the channel is active. |

### 15.7.10 static void EDMA\_SetChannelMux ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **int32\_t** *channelRequestSource* ) [inline], [static]

Parameters

|                             |                                                                                                                                                                                                                                                                                        |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>                 | eDMA peripheral base address.                                                                                                                                                                                                                                                          |
| <i>channel</i>              | eDMA channel number.                                                                                                                                                                                                                                                                   |
| <i>channelRequestSource</i> | eDMA hardware service request source for the channel. User need to use the <code>dma_request_source_t</code> type as the input parameter. Note that devices may use other enum type to express dma request source and User can fined it in SOC header or <code>fsl_edma_soc.h</code> . |

### 15.7.11 static **uint32\_t** EDMA\_GetChannelSystemBusInformation ( **EDMA\_Type** \* *base*, **uint32\_t** *channel* ) [inline], [static]

Parameters

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

Returns

The mask of the channel system bus information. Users need to use the `_edma_channel_sys_bus_info` type to decode the return variables.

### 15.7.12 static void EDMA\_EnableChannelMasterIDReplication ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **bool** *enable* ) [inline], [static]

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>base</i>    | eDMA peripheral base address.     |
| <i>channel</i> | eDMA channel number.              |
| <i>enable</i>  | true is enable, false is disable. |

**15.7.13 static void EDMA\_SetChannelSecurityLevel ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **edma\_channel\_security\_level\_t** *level* ) [inline], [static]**

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |
| <i>level</i>   | security level.               |

**15.7.14 static void EDMA\_SetChannelProtectionLevel ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **edma\_channel\_protection\_level\_t** *level* ) [inline], [static]**

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |
| <i>level</i>   | security level.               |

**15.7.15 void EDMA\_ResetChannel ( **EDMA\_Type** \* *base*, **uint32\_t** *channel* )**

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

Parameters

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

## Note

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

This function enables the auto stop request feature.

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

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

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

## Parameters

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

## Note

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

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

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

Parameters

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

### **15.7.18 void EDMA\_SetChannelPreemptionConfig ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **const edma\_channel\_Preemption\_config\_t** \* *config* )**

This function configures the channel preemption attribute and the priority of the channel.

Parameters

|                |                                                              |
|----------------|--------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                |
| <i>channel</i> | eDMA channel number                                          |
| <i>config</i>  | A pointer to the channel preemption configuration structure. |

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

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

Parameters

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

Note

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

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

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

Parameters

|                  |                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | eDMA peripheral base address.                                                                                                                                                                                 |
| <i>channel</i>   | eDMA channel number.                                                                                                                                                                                          |
| <i>bandWidth</i> | A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"> <li>• kEDMABandwidthStallNone</li> <li>• kEDMABandwidthStall4Cycle</li> <li>• kEDMABandwidthStall8Cycle</li> </ul> |

### 15.7.21 void EDMA\_SetModulo ( ***EDMA\_Type*** \* *base*, ***uint32\_t*** *channel*, ***edma\_modulo\_t*** *srcModulo*, ***edma\_modulo\_t*** *destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | eDMA peripheral base address. |
| <i>channel</i>    | eDMA channel number.          |
| <i>srcModulo</i>  | A source modulo value.        |
| <i>destModulo</i> | A destination modulo value.   |

### 15.7.22 static void EDMA\_EnableAutoStopRequest ( ***EDMA\_Type*** \* *base*, ***uint32\_t*** *channel*, ***bool*** *enable* ) [inline], [static]

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                    |
| <i>channel</i> | eDMA channel number.                             |
| <i>enable</i>  | The command to enable (true) or disable (false). |

### 15.7.23 void EDMA\_EnableChannelInterrupts ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **uint32\_t** *mask* )

Parameters

|                |                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                       |
| <i>channel</i> | eDMA channel number.                                                                                |
| <i>mask</i>    | The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type. |

### 15.7.24 void EDMA\_DisableChannelInterrupts ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **uint32\_t** *mask* )

Parameters

|                |                                                                                           |
|----------------|-------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                             |
| <i>channel</i> | eDMA channel number.                                                                      |
| <i>mask</i>    | The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type. |

### 15.7.25 void EDMA\_SetMajorOffsetConfig ( **EDMA\_Type** \* *base*, **uint32\_t** *channel*, **int32\_t** *sourceOffset*, **int32\_t** *destOffset* )

Adjustment value added to the source address at the completion of the major iteration count

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>base</i>         | eDMA peripheral base address.                                                       |
| <i>channel</i>      | edma channel number.                                                                |
| <i>sourceOffset</i> | source address offset will be applied to source address after major loop done.      |
| <i>destOffset</i>   | destination address offset will be applied to source address after major loop done. |

### 15.7.26 void EDMA\_ConfigChannelSoftwareTCD ( *edma\_tcd\_t* \* *tcd*, *const edma\_transfer\_config\_t* \* *transfer* )

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>tcd</i>      | Pointer to the TCD structure.           |
| <i>transfer</i> | channel transfer configuration pointer. |

Note

This function enables the auto stop request feature.

### 15.7.27 void EDMA\_TcdReset ( *edma\_tcd\_t* \* *tcd* )

This function sets all fields for this TCD structure to default value.

Parameters

|            |                               |
|------------|-------------------------------|
| <i>tcd</i> | Pointer to the TCD structure. |
|------------|-------------------------------|

Note

This function enables the auto stop request feature.

### 15.7.28 void EDMA\_TcdSetTransferConfig ( *edma\_tcd\_t* \* *tcd*, *const edma\_transfer\_config\_t* \* *config*, *edma\_tcd\_t* \* *nextTcd* )

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
* edma_transfer_t config = {
* ...
* }
* edma_tcd_t tcd __aligned(32);
* edma_tcd_t nextTcd __aligned(32);
* EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
*
```

Parameters

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| <i>tcd</i>     | Pointer to the TCD structure.                                                                            |
| <i>config</i>  | Pointer to eDMA transfer configuration structure.                                                        |
| <i>nextTcd</i> | Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature. |

Note

TCD address should be 32 bytes aligned or it causes an eDMA error.

If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA\_TcdReset.

### 15.7.29 void EDMA\_TcdSetMinorOffsetConfig ( *edma\_tcd\_t \* tcd, const edma\_minor\_offset\_config\_t \* config* )

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>tcd</i>    | A point to the TCD structure.                          |
| <i>config</i> | A pointer to the minor offset configuration structure. |

### 15.7.30 void EDMA\_TcdSetChannelLink ( *edma\_tcd\_t \* tcd, edma\_channel\_link\_type\_t type, uint32\_t linkedChannel* )

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

Note

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

Parameters

|                      |                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>           | Point to the TCD structure.                                                                                                                               |
| <i>type</i>          | Channel link type, it can be one of: <ul style="list-style-type: none"><li>• kEDMA_LinkNone</li><li>• kEDMA_MinorLink</li><li>• kEDMA_MajorLink</li></ul> |
| <i>linkedChannel</i> | The linked channel number.                                                                                                                                |

### 15.7.31 static void EDMA\_TcdSetBandWidth ( *edma\_tcd\_t \* tcd*, *edma\_bandwidth\_t bandWidth* ) [inline], [static]

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

Parameters

|                  |                                                                                                                                                                                                           |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>       | A pointer to the TCD structure.                                                                                                                                                                           |
| <i>bandWidth</i> | A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"><li>• kEDMABandwidthStallNone</li><li>• kEDMABandwidthStall4Cycle</li><li>• kEDMABandwidthStall8Cycle</li></ul> |

### 15.7.32 void EDMA\_TcdSetModulo ( *edma\_tcd\_t \* tcd*, *edma\_modulo\_t srcModulo*, *edma\_modulo\_t destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

|            |                                 |
|------------|---------------------------------|
| <i>tcd</i> | A pointer to the TCD structure. |
|------------|---------------------------------|

|                   |                             |
|-------------------|-----------------------------|
| <i>srcModulo</i>  | A source modulo value.      |
| <i>destModulo</i> | A destination modulo value. |

### 15.7.33 static void EDMA\_TcdEnableAutoStopRequest ( *edma\_tcd\_t* \* *tcd*, *bool enable* ) [inline], [static]

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>tcd</i>    | A pointer to the TCD structure.                  |
| <i>enable</i> | The command to enable (true) or disable (false). |

### 15.7.34 void EDMA\_TcdEnableInterrupts ( *edma\_tcd\_t* \* *tcd*, *uint32\_t mask* )

Parameters

|             |                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>  | Point to the TCD structure.                                                                                |
| <i>mask</i> | The mask of interrupt source to be set. Users need to use the defined <i>edma_interrupt_enable_t</i> type. |

### 15.7.35 void EDMA\_TcdDisableInterrupts ( *edma\_tcd\_t* \* *tcd*, *uint32\_t mask* )

Parameters

|             |                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>  | Point to the TCD structure.                                                                                |
| <i>mask</i> | The mask of interrupt source to be set. Users need to use the defined <i>edma_interrupt_enable_t</i> type. |

### 15.7.36 void EDMA\_TcdSetMajorOffsetConfig ( *edma\_tcd\_t* \* *tcd*, *int32\_t sourceOffset*, *int32\_t destOffset* )

Adjustment value added to the source address at the completion of the major iteration count

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>tcd</i>          | A point to the TCD structure.                                                       |
| <i>sourceOffset</i> | source address offset will be applied to source address after major loop done.      |
| <i>destOffset</i>   | destination address offset will be applied to source address after major loop done. |

### **15.7.37 static void EDMA\_EnableChannelRequest ( EDMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function enables the hardware channel request.

Parameters

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

### **15.7.38 static void EDMA\_DisableChannelRequest ( EDMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function disables the hardware channel request.

Parameters

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

### **15.7.39 static void EDMA\_TriggerChannelStart ( EDMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function starts a minor loop transfer.

Parameters

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

|                |                      |
|----------------|----------------------|
| <i>channel</i> | eDMA channel number. |
|----------------|----------------------|

### 15.7.40 **uint32\_t EDMA\_GetRemainingMajorLoopCount ( EDMA\_Type \* *base*, uint32\_t *channel* )**

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Parameters

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

Returns

Major loop count which has not been transferred yet for the current TCD.

Note

1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.
1. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA\_TCDn\_NBYTES\_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma\_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount \* NBYTE-S(initially configured)

### 15.7.41 **static uint32\_t EDMA\_GetErrorStatusFlags ( EDMA\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The mask of error status flags. Users need to use the \_edma\_error\_status\_flags type to decode the return variables.

#### 15.7.42 **uint32\_t EDMA\_GetChannelStatusFlags ( EDMA\_Type \* *base*, uint32\_t *channel* )**

Parameters

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

Returns

The mask of channel status flags. Users need to use the \_edma\_channel\_status\_flags type to decode the return variables.

#### 15.7.43 **void EDMA\_ClearChannelStatusFlags ( EDMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                            |
| <i>channel</i> | eDMA channel number.                                                                                     |
| <i>mask</i>    | The mask of channel status to be cleared. Users need to use the defined _edma_channel_status_flags type. |

#### 15.7.44 **void EDMA\_CreateHandle ( edma\_handle\_t \* *handle*, EDMA\_Type \* *base*, uint32\_t *channel* )**

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>handle</i>  | eDMA handle pointer. The eDMA handle stores callback function and parameters. |
| <i>base</i>    | eDMA peripheral base address.                                                 |
| <i>channel</i> | eDMA channel number.                                                          |

#### **15.7.45 void EDMA\_InstallTCDMemory( edma\_handle\_t \* *handle*, edma\_tcd\_t \* *tcdPool*, uint32\_t *tcdSize* )**

This function is called after the EDMA\_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA\_SubmitTransfer.

Parameters

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>handle</i>  | eDMA handle pointer.                                      |
| <i>tcdPool</i> | A memory pool to store TCDs. It must be 32 bytes aligned. |
| <i>tcdSize</i> | The number of TCD slots.                                  |

#### **15.7.46 void EDMA\_SetCallback( edma\_handle\_t \* *handle*, edma\_callback *callback*, void \* *userData* )**

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>handle</i>   | eDMA handle pointer.                   |
| <i>callback</i> | eDMA callback function pointer.        |
| <i>userData</i> | A parameter for the callback function. |

**15.7.47 void EDMA\_PrepTransferConfig ( *edma\_transfer\_config\_t* \* *config*,  
void \* *srcAddr*, *uint32\_t* *srcWidth*, *int16\_t* *srcOffset*, void \* *destAddr*,  
*uint32\_t* *destWidth*, *int16\_t* *destOffset*, *uint32\_t* *bytesEachRequest*,  
*uint32\_t* *transferBytes* )**

This function prepares the transfer configuration structure according to the user input.

## Parameters

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type edma_transfer_t. |
| <i>srcAddr</i>          | eDMA transfer source address.                             |
| <i>srcWidth</i>         | eDMA transfer source address width(bytes).                |
| <i>srcOffset</i>        | source address offset.                                    |
| <i>destAddr</i>         | eDMA transfer destination address.                        |
| <i>destWidth</i>        | eDMA transfer destination address width(bytes).           |
| <i>destOffset</i>       | destination address offset.                               |
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request.                  |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.                    |

## Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE). User can check if 128 bytes support is available for specific instance by FSL\_FEATURE\_EDMA\_INSTANCE\_SUPPORT\_128\_BYTES\_TRANSFERn.

#### 15.7.48 void EDMA\_PreparesTransfer ( edma\_transfer\_config\_t \* *config*, void \* *srcAddr*, uint32\_t *srcWidth*, void \* *destAddr*, uint32\_t *destWidth*, uint32\_t *bytesEachRequest*, uint32\_t *transferBytes*, edma\_transfer\_type\_t *type* )

This function prepares the transfer configuration structure according to the user input.

## Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>config</i>    | The user configuration structure of type edma_transfer_t. |
| <i>srcAddr</i>   | eDMA transfer source address.                             |
| <i>srcWidth</i>  | eDMA transfer source address width(bytes).                |
| <i>destAddr</i>  | eDMA transfer destination address.                        |
| <i>destWidth</i> | eDMA transfer destination address width(bytes).           |

|                         |                                          |
|-------------------------|------------------------------------------|
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request. |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.   |
| <i>type</i>             | eDMA transfer type.                      |

## Note

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

**15.7.49 void EDMA\_PreparesTransferTCD ( *edma\_handle\_t \* handle*, *edma\_tcd\_t \* tcd*, *void \* srcAddr*, *uint32\_t srcWidth*, *int16\_t srcOffset*, *void \* destAddr*, *uint32\_t destWidth*, *int16\_t destOffset*, *uint32\_t bytesEachRequest*, *uint32\_t transferBytes*, *edma\_tcd\_t \* nextTcd* )**

This function prepares the transfer content descriptor structure according to the user input.

## Parameters

|                         |                                                        |
|-------------------------|--------------------------------------------------------|
| <i>handle</i>           | eDMA handle pointer.                                   |
| <i>tcd</i>              | Pointer to eDMA transfer content descriptor structure. |
| <i>srcAddr</i>          | eDMA transfer source address.                          |
| <i>srcWidth</i>         | eDMA transfer source address width(bytes).             |
| <i>srcOffset</i>        | source address offset.                                 |
| <i>destAddr</i>         | eDMA transfer destination address.                     |
| <i>destWidth</i>        | eDMA transfer destination address width(bytes).        |
| <i>destOffset</i>       | destination address offset.                            |
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request.               |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.                 |
| <i>nextTcd</i>          | eDMA transfer linked TCD address.                      |

## Note

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

### 15.7.50 status\_t EDMA\_SubmitTransferTCD ( *edma\_handle\_t \* handle, const edma\_tcd\_t \* tcd* )

This function submits the eDMA transfer request according to the transfer content descriptor. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCDMemory before.

Typical user case:

1. submit single transfer

```
* edma_tcd_t tcd;
* EDMA_PrepareTransferTCD(handle, tcd,)
* EDMA_SubmitTransferTCD(handle, tcd)
* EDMA_StartTransfer(handle)
*
```

2. submit static link transfer,

```
* edma_tcd_t tcd[2];
* EDMA_PrepareTransferTCD(handle, &tcd[0],)
* EDMA_PrepareTransferTCD(handle, &tcd[1],)
* EDMA_SubmitTransferTCD(handle, &tcd[0])
* EDMA_StartTransfer(handle)
*
```

3. submit dynamic link transfer

```
* edma_tcd_t tcdpool[2];
* EDMA_InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
* edma_tcd_t tcd;
* EDMA_PrepareTransferTCD(handle, tcd,)
* EDMA_SubmitTransferTCD(handle, tcd)
* EDMA_PrepareTransferTCD(handle, tcd,)
* EDMA_SubmitTransferTCD(handle, tcd)
* EDMA_StartTransfer(handle)
*
```

4. submit loop transfer

```
* edma_tcd_t tcd[2];
* EDMA_PrepareTransferTCD(handle, &tcd[0], ..., &tcd[1])
* EDMA_PrepareTransferTCD(handle, &tcd[1], ..., &tcd[0])
* EDMA_SubmitTransferTCD(handle, &tcd[0])
* EDMA_StartTransfer(handle)
*
```

Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | eDMA handle pointer.                                   |
| <i>tcd</i>    | Pointer to eDMA transfer content descriptor structure. |

Return values

|                                |                                                                     |
|--------------------------------|---------------------------------------------------------------------|
| <i>kStatus_EDMA_Success</i>    | It means submit transfer request succeed.                           |
| <i>kStatus_EDMA_Queue-Full</i> | It means TCD queue is full. Submit transfer request is not allowed. |
| <i>kStatus_EDMA_Busy</i>       | It means the given channel is busy, need to submit request later.   |

### 15.7.51 status\_t EDMA\_SubmitTransfer ( *edma\_handle\_t \* handle*, *const edma\_transfer\_config\_t \* config* )

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA\_InstallTCDMemory before.

Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>handle</i> | eDMA handle pointer.                              |
| <i>config</i> | Pointer to eDMA transfer configuration structure. |

Return values

|                                |                                                                     |
|--------------------------------|---------------------------------------------------------------------|
| <i>kStatus_EDMA_Success</i>    | It means submit transfer request succeed.                           |
| <i>kStatus_EDMA_Queue-Full</i> | It means TCD queue is full. Submit transfer request is not allowed. |
| <i>kStatus_EDMA_Busy</i>       | It means the given channel is busy, need to submit request later.   |

### 15.7.52 status\_t EDMA\_SubmitLoopTransfer ( *edma\_handle\_t \* handle*, *edma\_transfer\_config\_t \* transfer*, *uint32\_t transferLoopCount* )

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
* edma_channel_transfer_config_t transfer[2];
* EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
*
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

## Note

Application should check the return value of this function to avoid transfer request submit failed

## Parameters

|                           |                                                                                                                |
|---------------------------|----------------------------------------------------------------------------------------------------------------|
| <i>handle</i>             | eDMA handle pointer                                                                                            |
| <i>transfer</i>           | pointer to user's eDMA channel configure structure, see <code>edma_channel_transfer_config_t</code> for detail |
| <i>transferLoop-Count</i> | the count of the transfer ring, if loop count is 1, that means that the one will link to itself.               |

## Return values

|                                |                                          |
|--------------------------------|------------------------------------------|
| <i>kStatus_Success</i>         | It means submit transfer request succeed |
| <i>kStatus_EDMA_Busy</i>       | channel is in busy status                |
| <i>kStatus_InvalidArgument</i> | Invalid Argument                         |

**15.7.53 void EDMA\_StartTransfer( `edma_handle_t * handle` )**

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | eDMA handle pointer. |
|---------------|----------------------|

**15.7.54 void EDMA\_StopTransfer( `edma_handle_t * handle` )**

This function disables the channel request to pause the transfer. Users can call `EDMA_StartTransfer()` again to resume the transfer.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | eDMA handle pointer. |
|---------------|----------------------|

**15.7.55 void EDMA\_AbortTransfer( `edma_handle_t * handle` )**

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

### 15.7.56 static uint32\_t EDMA\_GetUnusedTCDNumber ( *edma\_handle\_t \* handle* ) [**inline**], [**static**]

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

Returns

The unused tcd slot number.

### 15.7.57 static uint32\_t EDMA\_GetNextTCDAccount ( *edma\_handle\_t \* handle* ) [**inline**], [**static**]

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

Returns

The next TCD address.

### 15.7.58 void EDMA\_HandleIRQ ( *edma\_handle\_t \* handle* )

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga\_index are calculated based on the DLAST\_SGA bitfield lies in the TCD\_CSR register, the sga\_index in this case should be 2 (DLAST\_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

### Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | eDMA handle pointer. |
|---------------|----------------------|

## 15.8 eDMA core Driver

## 15.9 eDMA soc Driver

### 15.9.1 Overview

#### Macros

- `#define FSL_EDMA_SOC_IP_DMA3 (1)`  
*DMA IP version.*
- `#define EDMA_BASE_PTRS`  
*DMA base table.*
- `#define EDMA_CHN IRQS`  
*Verify dma base and request source.*
- `#define EDMA_CHANNEL_OFFSET 0x10000U`  
*EDMA base address convert macro.*

#### Driver version

- `#define FSL_EDMA_SOC_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`  
*Driver version 2.1.0.*

### 15.9.2 Macro Definition Documentation

#### 15.9.2.1 `#define FSL_EDMA_SOC_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

# Chapter 16

## FlexCAN: Flex Controller Area Network Driver

### 16.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flex Controller Area Network (FlexCAN) module of MCUXpresso SDK devices.

### Modules

- [FlexCAN Driver](#)

## 16.2 FlexCAN Driver

### 16.2.1 Overview

This section describes the programming interface of the FlexCAN driver. The FlexCAN driver configures FlexCAN module and provides functional and transactional interfaces to build the FlexCAN application.

### 16.2.2 Typical use case

#### 16.2.2.1 Message Buffer Send Operation

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

#### 16.2.2.2 Message Buffer Receive Operation

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

#### 16.2.2.3 Receive FIFO Operation

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

## Data Structures

- struct [flexcan\\_memory\\_error\\_report\\_status\\_t](#)  
*FlexCAN memory error register status structure. [More...](#)*
- struct [flexcan\\_frame\\_t](#)  
*FlexCAN message frame structure. [More...](#)*
- struct [flexcan\\_fd\\_frame\\_t](#)  
*CAN FD message frame structure. [More...](#)*
- struct [flexcan\\_timing\\_config\\_t](#)  
*FlexCAN protocol timing characteristic configuration structure. [More...](#)*
- struct [flexcan\\_config\\_t](#)  
*FlexCAN module configuration structure. [More...](#)*
- struct [flexcan\\_rx\\_mb\\_config\\_t](#)  
*FlexCAN Receive Message Buffer configuration structure. [More...](#)*
- struct [flexcan\\_rx\\_fifo\\_config\\_t](#)  
*FlexCAN Legacy Rx FIFO configuration structure. [More...](#)*
- struct [flexcan\\_enhanced\\_rx\\_fifo\\_std\\_id\\_filter\\_t](#)  
*FlexCAN Enhanced Rx FIFO Standard ID filter element structure. [More...](#)*
- struct [flexcan\\_enhanced\\_rx\\_fifo\\_ext\\_id\\_filter\\_t](#)  
*FlexCAN Enhanced Rx FIFO Extended ID filter element structure. [More...](#)*
- struct [flexcan\\_enhanced\\_rx\\_fifo\\_config\\_t](#)  
*FlexCAN Enhanced Rx FIFO configuration structure. [More...](#)*

- struct **flexcan\_mb\_transfer\_t**  
*FlexCAN Message Buffer transfer.* [More...](#)
- struct **flexcan\_fifo\_transfer\_t**  
*FlexCAN Rx FIFO transfer.* [More...](#)
- struct **flexcan\_handle\_t**  
*FlexCAN handle structure.* [More...](#)

## Macros

- #define **DLC\_LENGTH\_DECODE**(dlc) (((dlc) <= 8U) ? (dlc) : (((dlc) <= 12U) ? (((dlc)-6U) \* 4U) : (((dlc)-11U) \* 16U)))  
*FlexCAN frame length helper macro.*
- #define **FLEXCAN\_ID\_STD**(id) (((uint32\_t)((uint32\_t)(id)) << CAN\_ID\_STD\_SHIFT)) & CAN\_ID\_STD\_MASK)  
*FlexCAN Frame ID helper macro.*
- #define **FLEXCAN\_ID\_EXT**(id)  
*Extend Frame ID helper macro.*
- #define **FLEXCAN\_RX\_MB\_STD\_MASK**(id, rtr, ide)  
*FlexCAN Rx Message Buffer Mask helper macro.*
- #define **FLEXCAN\_RX\_MB\_EXT\_MASK**(id, rtr, ide)  
*Extend Rx Message Buffer Mask helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A**(id, rtr, ide)  
*FlexCAN Legacy Rx FIFO Mask helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH**(id, rtr, ide)  
*Standard Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW**(id, rtr, ide)  
*Standard Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH**(id) (((uint32\_t)(id)&0x7F8) << 21)  
*Standard Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH**(id) (((uint32\_t)(id)&0x7F8) << 13)  
*Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW**(id) (((uint32\_t)(id)&0x7F8) << 5)  
*Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW**(id) (((uint32\_t)(id)&0x7F8) >> 3)  
*Standard Rx FIFO Mask helper macro Type C lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A**(id, rtr, ide)  
*Extend Rx FIFO Mask helper macro Type A helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH**(id, rtr, ide)  
*Extend Rx FIFO Mask helper macro Type B upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW**(id, rtr, ide)  
*Extend Rx FIFO Mask helper macro Type B lower part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH**(id) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) << 3)  
*Extend Rx FIFO Mask helper macro Type C upper part helper macro.*
- #define **FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH**(id)  
*Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.*

- `#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(id)`  
*Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)`  
*Extend Rx FIFO Mask helper macro Type C lower part helper macro.*
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_STD_MASK_TYPE_A(id, rtr, ide)`  
*FlexCAN Rx FIFO Filter helper macro.*
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(id, rtr, ide)`  
*Standard Rx FIFO Filter helper macro Type B upper part helper macro.*
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(id, rtr, ide)`  
*Standard Rx FIFO Filter helper macro Type B lower part helper macro.*
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(id)`  
*Standard Rx FIFO Filter helper macro Type C upper part helper macro.*
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(id)`  
*Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.*
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(id)`  
*Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.*
- `#define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(id)`  
*Standard Rx FIFO Filter helper macro Type C lower part helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(id, rtr, ide) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)`  
*Extend Rx FIFO Filter helper macro Type A helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(id, rtr, ide)`  
*Extend Rx FIFO Filter helper macro Type B upper part helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(id, rtr, ide)`  
*Extend Rx FIFO Filter helper macro Type B lower part helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(id)`  
*Extend Rx FIFO Filter helper macro Type C upper part helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(id)`  
*Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(id)`  
*Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.*
- `#define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)`  
*Extend Rx FIFO Filter helper macro Type C lower part helper macro.*
- `#define ENHANCED_RX_FIFO_FSCH(x) (((uint32_t)((uint32_t)(x) << 30)) & 0xC0000000U)`  
*FlexCAN Enhanced Rx FIFO Filter and Mask helper macro.*
- `#define FLEXCAN_ENHANCED_RX_FIFO_STD_MASK_AND_FILTER(id, rtr, id_mask, rtr_mask) (ENHANCED_RX_FIFO_FSCH(0x0) | RTR_STD_HIGH(rtr) | ID_STD_HIGH(id) | RTR_STD_LOW(rtr_mask) | ID_STD_LOW(id_mask))`  
*Standard ID filter element with filter + mask scheme.*
- `#define FLEXCAN_ENHANCED_RX_FIFO_STD_FILTER_WITH_RANGE(id_upper, rtr, id_lower, rtr_mask)`  
*Standard ID filter element with filter range.*
- `#define FLEXCAN_ENHANCED_RX_FIFO_STD_TWO_FILTERS(id1, rtr1, id2, rtr2) (ENHANCED_RX_FIFO_FSCH(0x2) | RTR_STD_HIGH(rtr1) | ID_STD_HIGH(id1) | RTR_STD_LOW(rtr2) | ID_STD_LOW(id2))`  
*Standard ID filter element with two filters without masks.*
- `#define FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_LOW(id, rtr) (ENH-`

`ANCED_RX_FIFO_FSCH(0x0) | RTR_EXT(rtr) | ID_EXT(id))`

*Extended ID filter element with filter + mask scheme low word.*

- `#define FLEXCAN_ENHANCED_RX_FIFO_EXT_MASK_AND_FILTER_HIGH(id_mask, rtr_mask) (ENHANCED_RX_FIFO_FSCH(0x0) | RTR_EXT(rtr_mask) | ID_EXT(id_mask))`

*Extended ID filter element with filter + mask scheme high word.*

- `#define FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_LOW(id_upper, rtr) (ENHANCED_RX_FIFO_FSCH(0x1) | RTR_EXT(rtr) | ID_EXT(id_upper))`

*Extended ID filter element with range scheme low word.*

- `#define FLEXCAN_ENHANCED_RX_FIFO_EXT_FILTER_WITH_RANGE_HIGH(id_lower, rtr_mask) (ENHANCED_RX_FIFO_FSCH(0x1) | RTR_EXT(rtr_mask) | ID_EXT(id_lower))`

*Extended ID filter element with range scheme high word.*

- `#define FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_LOW(id2, rtr2) (ENHANCED_RX_FIFO_FSCH(0x2) | RTR_EXT(rtr2) | ID_EXT(id2))`

*Extended ID filter element with two filters without masks low word.*

- `#define FLEXCAN_ENHANCED_RX_FIFO_EXT_TWO_FILTERS_HIGH(id1, rtr1) (ENHANCED_RX_FIFO_FSCH(0x2) | RTR_EXT(rtr1) | ID_EXT(id1))`

*Extended ID filter element with two filters without masks high word.*

- `#define FLEXCAN_EFIFO_INT_MASK(x) (((uint64_t)((uint64_t)(x) << 32)) & 0xF000000000000000U)`

*FlexCAN interrupt/status flag helper macro.*

- `#define E_RX_FIFO(base) ((uintptr_t)(base) + 0x2000U)`

*FlexCAN Enhanced Rx FIFO base address helper macro.*

- `#define FLEXCAN_CALLBACK(x) void(x)(CAN_Type * base, flexcan_handle_t * handle, status_t status, uint64_t result, void *userData)`

*FlexCAN transfer callback function.*

## Enumerations

- enum {
 `kStatus_FLEXCAN_TxBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 0),`
`kStatus_FLEXCAN_TxIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 1),`
`kStatus_FLEXCAN_TxSwitchToRx,`
`kStatus_FLEXCAN_RxBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 3),`
`kStatus_FLEXCAN_RxIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 4),`
`kStatus_FLEXCAN_RxOverflow = MAKE_STATUS(kStatusGroup_FLEXCAN, 5),`
`kStatus_FLEXCAN_RxFifoBusy = MAKE_STATUS(kStatusGroup_FLEXCAN, 6),`
`kStatus_FLEXCAN_RxFifoIdle = MAKE_STATUS(kStatusGroup_FLEXCAN, 7),`
`kStatus_FLEXCAN_RxFifoOverflow = MAKE_STATUS(kStatusGroup_FLEXCAN, 8),`
`kStatus_FLEXCAN_RxFifoWarning = MAKE_STATUS(kStatusGroup_FLEXCAN, 9),`
`kStatus_FLEXCAN_RxFifoDisabled,`
`kStatus_FLEXCAN_ErrorStatus = MAKE_STATUS(kStatusGroup_FLEXCAN, 11),`
`kStatus_FLEXCAN_WakeUp = MAKE_STATUS(kStatusGroup_FLEXCAN, 12),`
`kStatus_FLEXCAN_UnHandled = MAKE_STATUS(kStatusGroup_FLEXCAN, 13),`
`kStatus_FLEXCAN_RxRemote = MAKE_STATUS(kStatusGroup_FLEXCAN, 14),`
`kStatus_FLEXCAN_RxFifoUnderflow }`

*FlexCAN transfer status.*

- enum `flexcan_frame_format_t` {
   
  `kFLEXCAN_FrameFormatStandard` = 0x0U,
   
  `kFLEXCAN_FrameFormatExtend` = 0x1U }
   
*FlexCAN frame format.*
- enum `flexcan_frame_type_t` {
   
  `kFLEXCAN_FrameTypeData` = 0x0U,
   
  `kFLEXCAN_FrameTypeRemote` = 0x1U }
   
*FlexCAN frame type.*
- enum `flexcan_clock_source_t` {
   
  `kFLEXCAN_ClkSrcOsc` = 0x0U,
   
  `kFLEXCAN_ClkSrcPeri` = 0x1U,
   
  `kFLEXCAN_ClkSrc0` = 0x0U,
   
  `kFLEXCAN_ClkSrc1` = 0x1U }
   
*FlexCAN clock source.*
- enum `flexcan_wake_up_source_t` {
   
  `kFLEXCAN_WakeupSrcUnfiltered` = 0x0U,
   
  `kFLEXCAN_WakeupSrcFiltered` = 0x1U }
   
*FlexCAN wake up source.*
- enum `flexcan_rx_fifo_filter_type_t` {
   
  `kFLEXCAN_RxFifoFilterTypeA` = 0x0U,
   
  `kFLEXCAN_RxFifoFilterTypeB`,
   
  `kFLEXCAN_RxFifoFilterTypeC`,
   
  `kFLEXCAN_RxFifoFilterTypeD` = 0x3U }
   
*FlexCAN Rx Fifo Filter type.*
- enum `flexcan_mb_size_t` {
   
  `kFLEXCAN_8BperMB` = 0x0U,
   
  `kFLEXCAN_16BperMB` = 0x1U,
   
  `kFLEXCAN_32BperMB` = 0x2U,
   
  `kFLEXCAN_64BperMB` = 0x3U }
   
*FlexCAN Message Buffer Payload size.*
- enum `_flexcan_fd_frame_length` {
   
  `kFLEXCAN_0BperFrame` = 0x0U,
   
  `kFLEXCAN_1BperFrame`,
   
  `kFLEXCAN_2BperFrame`,
   
  `kFLEXCAN_3BperFrame`,
   
  `kFLEXCAN_4BperFrame`,
   
  `kFLEXCAN_5BperFrame`,
   
  `kFLEXCAN_6BperFrame`,
   
  `kFLEXCAN_7BperFrame`,
   
  `kFLEXCAN_8BperFrame`,
   
  `kFLEXCAN_12BperFrame`,
   
  `kFLEXCAN_16BperFrame`,
   
  `kFLEXCAN_20BperFrame`,
   
  `kFLEXCAN_24BperFrame`,
   
  `kFLEXCAN_32BperFrame`,
   
  `kFLEXCAN_48BperFrame`,

```
kFLEXCAN_64BperFrame }
```

*FlexCAN CAN FD frame supporting data length (available DLC values).*

- enum `flexcan_efifo_dma_per_read_length_t` {
 `kFLEXCAN_1WordPerRead` = 0x0U,
 `kFLEXCAN_2WordPerRead`,
 `kFLEXCAN_3WordPerRead`,
 `kFLEXCAN_4WordPerRead`,
 `kFLEXCAN_5WordPerRead`,
 `kFLEXCAN_6WordPerRead`,
 `kFLEXCAN_7WordPerRead`,
 `kFLEXCAN_8WordPerRead`,
 `kFLEXCAN_9WordPerRead`,
 `kFLEXCAN_10WordPerRead`,
 `kFLEXCAN_11WordPerRead`,
 `kFLEXCAN_12WordPerRead`,
 `kFLEXCAN_13WordPerRead`,
 `kFLEXCAN_14WordPerRead`,
 `kFLEXCAN_15WordPerRead`,
 `kFLEXCAN_16WordPerRead`,
 `kFLEXCAN_17WordPerRead`,
 `kFLEXCAN_18WordPerRead`,
 `kFLEXCAN_19WordPerRead` }

*FlexCAN Enhanced Rx Fifo DMA transfer per read length enumerations.*

- enum `flexcan_rx_fifo_priority_t` {
 `kFLEXCAN_RxFifoPrioLow` = 0x0U,
 `kFLEXCAN_RxFifoPrioHigh` = 0x1U }

*FlexCAN Enhanced/Legacy Rx FIFO priority.*

- enum `_flexcan_interrupt_enable` {
 `kFLEXCAN_BusOffInterruptEnable` = CAN\_CTRL1\_BOFFMSK\_MASK,
 `kFLEXCAN_ErrorInterruptEnable` = CAN\_CTRL1\_ERRMSK\_MASK,
 `kFLEXCAN_TxWarningInterruptEnable` = CAN\_CTRL1\_TWRNMSK\_MASK,
 `kFLEXCAN_RxWarningInterruptEnable` = CAN\_CTRL1\_RWRNMSK\_MASK,
 `kFLEXCAN_WakeUpInterruptEnable` = CAN\_MCR\_WAKMSK\_MASK,
 `kFLEXCAN_FDErrorInterruptEnable` = CAN\_CTRL2\_ERRMSK\_FAST\_MASK,
 `kFLEXCAN_ERxFifoUnderflowInterruptEnable` = FLEXCAN\_EFIFO\_INT\_MASK(CAN\_ERFI-ER\_ERFUFWIE\_MASK),
 `kFLEXCAN_ERxFifoOverflowInterruptEnable` = FLEXCAN\_EFIFO\_INT\_MASK(CAN\_ERFIE-R\_ERFOVFIE\_MASK),
 `kFLEXCAN_ERxFifoWatermarkInterruptEnable` = FLEXCAN\_EFIFO\_INT\_MASK(CAN\_ERFI-ER\_ERFWMIIE\_MASK),
 `kFLEXCAN_HostAccessNCErrorInterruptEnable` = FLEXCAN\_MEGR\_INT\_MASK(CAN\_ME-CR\_HANCEI\_MSK\_MASK),
 `kFLEXCAN_FlexCanAccessNCErrorInterruptEnable` = FLEXCAN\_MEGR\_INT\_MASK(CAN\_-MECR\_FANCEI\_MSK\_MASK),
 `kFLEXCAN_HostOrFlexCanCErrorInterruptEnable` = FLEXCAN\_MEGR\_INT\_MASK(CAN\_M-

```

 ECR_CE1_MSK_MASK) }

FlexCAN interrupt enable enumerations.
• enum _flexcan_flags {
 kFLEXCAN_ErrorOverrunFlag = CAN_ESR1_ERROVR_MASK,
 kFLEXCAN_FDErrorIntFlag = CAN_ESR1_ERRINT_FAST_MASK,
 kFLEXCAN_BusoffDoneIntFlag = CAN_ESR1_BOFFDONEINT_MASK,
 kFLEXCAN_SynchFlag = CAN_ESR1_SYNCH_MASK,
 kFLEXCAN_TxWarningIntFlag = CAN_ESR1_TWRNINT_MASK,
 kFLEXCAN_RxWarningIntFlag = CAN_ESR1_RWRNINT_MASK,
 kFLEXCAN_IdleFlag = CAN_ESR1_IDLE_MASK,
 kFLEXCAN_FaultConfinementFlag = CAN_ESR1_FLTCONF_MASK,
 kFLEXCAN_TransmittingFlag = CAN_ESR1_TX_MASK,
 kFLEXCAN_ReceivingFlag = CAN_ESR1_RX_MASK,
 kFLEXCAN_BusOffIntFlag = CAN_ESR1_BOFFINT_MASK,
 kFLEXCAN_ErrorIntFlag = CAN_ESR1_ERRINT_MASK,
 kFLEXCAN_WakeUpIntFlag = CAN_ESR1_WAKINT_MASK ,
 kFLEXCAN_ERxFifoUnderflowIntFlag,
 kFLEXCAN_ERxFifoOverflowIntFlag,
 kFLEXCAN_ERxFifoWatermarkIntFlag,
 kFLEXCAN_ERxFifoDataAvlIntFlag,
 kFLEXCAN_ERxFifoEmptyFlag = FLEXCAN_EFIFO_STATUS_MASK(CAN_ERFSR_ERFE-
 _MASK),
 kFLEXCAN_ERxFifoFullFlag = FLEXCAN_EFIFO_STATUS_MASK(CAN_ERFSR_ERFF_M-
 ASK),
 kFLEXCAN_HostAccessNonCorrectableErrorIntFlag = FLEXCAN_MEGR_INT_MASK(CAN_-
 ERRSR_HANCEIF_MASK),
 kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag = FLEXCAN_MEGR_INT_MASK(CA-
 N_ERRSR_FANCEIF_MASK),
 kFLEXCAN_CorrectableErrorIntFlag = FLEXCAN_MEGR_INT_MASK(CAN_ERRSR_CEIF_-
 MASK),
 kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag = FLEXCAN_MEGR_INT_MASK(C-
 AN_ERRSR_HANCEIOF_MASK),
 kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag = FLEXCAN_MEGR_INT_MAS-
 K(CAN_ERRSR_FANCEIOF_MASK),
 kFLEXCAN_CorrectableErrorOverrunFlag = FLEXCAN_MEGR_INT_MASK(CAN_ERRSR_C-
 EIOF_MASK),
 kFLEXCAN_AllMemoryErrorFlag }

FlexCAN status flags.
• enum _flexcan_error_flags {

```

```

kFLEXCAN_FDStuffingError = CAN_ESR1_STFERR_FAST_MASK,
kFLEXCAN_FDFormError = CAN_ESR1_FRMERR_FAST_MASK,
kFLEXCAN_FDCrcError = CAN_ESR1_CRCERR_FAST_MASK,
kFLEXCAN_FDBit0Error = CAN_ESR1_BIT0ERR_FAST_MASK,
kFLEXCAN_FDBit1Error = (int)CAN_ESR1_BIT1ERR_FAST_MASK,
kFLEXCAN_TxErrorWarningFlag = CAN_ESR1_TXWRN_MASK,
kFLEXCAN_RxErrorWarningFlag = CAN_ESR1_RXWRN_MASK,
kFLEXCAN_StuffingError = CAN_ESR1_STFERR_MASK,
kFLEXCAN_FormError = CAN_ESR1_FRMERR_MASK,
kFLEXCAN_CrcError = CAN_ESR1_CRCERR_MASK,
kFLEXCAN_AckError = CAN_ESR1_ACKERR_MASK,
kFLEXCAN_Bit0Error = CAN_ESR1_BIT0ERR_MASK,
kFLEXCAN_Bit1Error = CAN_ESR1_BIT1ERR_MASK }

```

*FlexCAN error status flags.*

- enum {
 kFLEXCAN\_RxFifoOverflowFlag = CAN\_IFLAG1\_BUF7I\_MASK,
 kFLEXCAN\_RxFifoWarningFlag = CAN\_IFLAG1\_BUF6I\_MASK,
 kFLEXCAN\_RxFifoFrameAvlFlag = CAN\_IFLAG1\_BUF5I\_MASK }

*FlexCAN Legacy Rx FIFO status flags.*

- enum `flexcan_memory_error_type_t` {
 kFLEXCAN\_CorrectableError = 0U,
 kFLEXCAN\_NonCorrectableError }
- enum `flexcan_memory_access_type_t` {
 kFLEXCAN\_MoveOutFlexCanAccess = 0U,
 kFLEXCAN\_MoveInAccess,
 kFLEXCAN\_TxArbitrationAccess,
 kFLEXCAN\_RxMatchingAccess,
 kFLEXCAN\_MoveOutHostAccess }

*FlexCAN Memory Access Type.*

- enum `flexcan_byte_error_syndrome_t` {

```

kFLEXCAN_NoError = 0U,
kFLEXCAN_ParityBits0Error = 1U,
kFLEXCAN_ParityBits1Error = 2U,
kFLEXCAN_ParityBits2Error = 4U,
kFLEXCAN_ParityBits3Error = 8U,
kFLEXCAN_ParityBits4Error = 16U,
kFLEXCAN_DataBits0Error = 28U,
kFLEXCAN_DataBits1Error = 22U,
kFLEXCAN_DataBits2Error = 19U,
kFLEXCAN_DataBits3Error = 25U,
kFLEXCAN_DataBits4Error = 26U,
kFLEXCAN_DataBits5Error = 7U,
kFLEXCAN_DataBits6Error = 21U,
kFLEXCAN_DataBits7Error = 14U,
kFLEXCAN_AllZeroError = 6U,
kFLEXCAN_AllOneError = 31U,
kFLEXCAN_NonCorrectableErrors }

```

*FlexCAN Memory Error Byte Syndrome.*

## Driver version

- #define **FSL\_FLEXCAN\_DRIVER\_VERSION** (MAKE\_VERSION(2, 11, 4))  
*FlexCAN driver version.*

## Initialization and deinitialization

- bool **FLEXCAN\_IsInstanceHasFDMode** (CAN\_Type \*base)  
*Determine whether the FlexCAN instance support CAN FD mode at run time.*
- void **FLEXCAN\_EnterFreezeMode** (CAN\_Type \*base)  
*Enter FlexCAN Freeze Mode.*
- void **FLEXCAN\_ExitFreezeMode** (CAN\_Type \*base)  
*Exit FlexCAN Freeze Mode.*
- uint32\_t **FLEXCANGetInstance** (CAN\_Type \*base)  
*Get the FlexCAN instance from peripheral base address.*
- bool **FLEXCAN\_CalculateImprovedTimingValues** (CAN\_Type \*base, uint32\_t bitRate, uint32\_t sourceClock\_Hz, **flexcan\_timing\_config\_t** \*pTimingConfig)  
*Calculates the improved timing values by specific bit Rates for classical CAN.*
- void **FLEXCAN\_Init** (CAN\_Type \*base, const **flexcan\_config\_t** \*pConfig, uint32\_t sourceClock\_Hz)  
*Initializes a FlexCAN instance.*
- bool **FLEXCAN\_FDCalculateImprovedTimingValues** (CAN\_Type \*base, uint32\_t bitRate, uint32\_t bitRateFD, uint32\_t sourceClock\_Hz, **flexcan\_timing\_config\_t** \*pTimingConfig)  
*Calculates the improved timing values by specific bit rates for CANFD.*
- void **FLEXCAN\_FDIInit** (CAN\_Type \*base, const **flexcan\_config\_t** \*pConfig, uint32\_t sourceClock\_Hz, **flexcan\_mb\_size\_t** dataSize, bool brs)

- **void FLEXCAN\_Deinit (CAN\_Type \*base)**  
*De-initializes a FlexCAN instance.*
- **void FLEXCAN\_GetDefaultConfig (flexcan\_config\_t \*pConfig)**  
*Gets the default configuration structure.*

## Configuration.

- **void FLEXCAN\_SetTimingConfig (CAN\_Type \*base, const flexcan\_timing\_config\_t \*pConfig)**  
*Sets the FlexCAN classical CAN protocol timing characteristic.*
- **status\_t FLEXCAN\_SetBitRate (CAN\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t bitRate\_Bps)**  
*Set bit rate of FlexCAN classical CAN frame or CAN FD frame nominal phase.*
- **void FLEXCAN\_SetFDTimingConfig (CAN\_Type \*base, const flexcan\_timing\_config\_t \*pConfig)**  
*Sets the FlexCAN CANFD data phase timing characteristic.*
- **status\_t FLEXCAN\_SetFDBitRate (CAN\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t bitRate\_N\_Bps, uint32\_t bitRateD\_Bps)**  
*Set bit rate of FlexCAN FD frame.*
- **void FLEXCAN\_SetRxMbGlobalMask (CAN\_Type \*base, uint32\_t mask)**  
*Sets the FlexCAN receive message buffer global mask.*
- **void FLEXCAN\_SetRxFifoGlobalMask (CAN\_Type \*base, uint32\_t mask)**  
*Sets the FlexCAN receive FIFO global mask.*
- **void FLEXCAN\_SetRxIndividualMask (CAN\_Type \*base, uint8\_t maskIdx, uint32\_t mask)**  
*Sets the FlexCAN receive individual mask.*
- **void FLEXCAN\_SetTxMbConfig (CAN\_Type \*base, uint8\_t mbIdx, bool enable)**  
*Configures a FlexCAN transmit message buffer.*
- **void FLEXCAN\_SetFDTxMbConfig (CAN\_Type \*base, uint8\_t mbIdx, bool enable)**  
*Configures a FlexCAN transmit message buffer.*
- **void FLEXCAN\_SetRxMbConfig (CAN\_Type \*base, uint8\_t mbIdx, const flexcan\_rx\_mb\_config\_t \*pRxMbConfig, bool enable)**  
*Configures a FlexCAN Receive Message Buffer.*
- **void FLEXCAN\_SetFDRxMbConfig (CAN\_Type \*base, uint8\_t mbIdx, const flexcan\_rx\_mb\_config\_t \*pRxMbConfig, bool enable)**  
*Configures a FlexCAN Receive Message Buffer.*
- **void FLEXCAN\_SetRxFifoConfig (CAN\_Type \*base, const flexcan\_rx\_fifo\_config\_t \*pRxFifoConfig, bool enable)**  
*Configures the FlexCAN Legacy Rx FIFO.*
- **void FLEXCAN\_SetEnhancedRxFifoConfig (CAN\_Type \*base, const flexcan\_enhanced\_rx\_fifo\_config\_t \*pConfig, bool enable)**  
*Configures the FlexCAN Enhanced Rx FIFO.*

## Status

- **static uint64\_t FLEXCAN\_GetStatusFlags (CAN\_Type \*base)**  
*Gets the FlexCAN module interrupt flags.*
- **static void FLEXCAN\_ClearStatusFlags (CAN\_Type \*base, uint64\_t mask)**  
*Clears status flags with the provided mask.*
- **static void FLEXCAN\_GetBusErrCount (CAN\_Type \*base, uint8\_t \*txErrBuf, uint8\_t \*rxErrBuf)**

- static uint64\_t **FLEXCAN\_GetMbStatusFlags** (CAN\_Type \*base, uint64\_t mask)
 

*Gets the FlexCAN Bus Error Counter value.*
- static uint64\_t **FLEXCAN\_GetHigh64MbStatusFlags** (CAN\_Type \*base, uint64\_t mask)
 

*Gets the FlexCAN Message Buffer interrupt flags.*
- static void **FLEXCAN\_ClearMbStatusFlags** (CAN\_Type \*base, uint64\_t mask)
 

*Clears the FlexCAN Message Buffer interrupt flags.*
- static void **FLEXCAN\_ClearHigh64MbStatusFlags** (CAN\_Type \*base, uint64\_t mask)
 

*Clears the FlexCAN High 64 Message Buffer interrupt flags.*
- void **FLEXCAN\_GetMemoryErrorReportStatus** (CAN\_Type \*base, **flexcan\_memory\_error\_report\_status\_t** \*errorStatus)
 

*Gets the FlexCAN Memory Error Report registers status.*
- static uint32\_t **FLEXCAN\_GetEnhancedFifoDataCount** (CAN\_Type \*base)
 

*Gets the number of FlexCAN Enhanced Rx FIFO available frames.*

## Interrupts

- static void **FLEXCAN\_EnableInterrupts** (CAN\_Type \*base, uint64\_t mask)
 

*Enables FlexCAN interrupts according to the provided mask.*
- static void **FLEXCAN\_DisableInterrupts** (CAN\_Type \*base, uint64\_t mask)
 

*Disables FlexCAN interrupts according to the provided mask.*
- static void **FLEXCAN\_EnableMbInterrupts** (CAN\_Type \*base, uint64\_t mask)
 

*Enables FlexCAN Message Buffer interrupts.*
- static void **FLEXCAN\_EnableHigh64MbInterrupts** (CAN\_Type \*base, uint64\_t mask)
 

*Enables FlexCAN high 64 Message Buffer interrupts.*
- static void **FLEXCAN\_DisableMbInterrupts** (CAN\_Type \*base, uint64\_t mask)
 

*Disables FlexCAN Message Buffer interrupts.*
- static void **FLEXCAN\_DisableHigh64MbInterrupts** (CAN\_Type \*base, uint64\_t mask)
 

*Disables FlexCAN high 64 Message Buffer interrupts.*

## DMA Control

- void **FLEXCAN\_EnableRx\_fifoDMA** (CAN\_Type \*base, bool enable)
 

*Enables or disables the FlexCAN Rx FIFO DMA request.*
- static uintptr\_t **FLEXCAN\_GetRx\_fifoHeadAddr** (CAN\_Type \*base)
 

*Gets the Rx FIFO Head address.*

## Bus Operations

- static void **FLEXCAN\_Enable** (CAN\_Type \*base, bool enable)
 

*Enables or disables the FlexCAN module operation.*
- **status\_t FLEXCAN\_WriteTxMb** (CAN\_Type \*base, uint8\_t mbIdx, const **flexcan\_frame\_t** \*pTxFrame)
 

*Writes a FlexCAN Message to the Transmit Message Buffer.*
- **status\_t FLEXCAN\_ReadRxMb** (CAN\_Type \*base, uint8\_t mbIdx, **flexcan\_frame\_t** \*pRxFrame)
 

*Reads a FlexCAN Message from Receive Message Buffer.*

- **status\_t FLEXCAN\_WriteFDTxMb** (CAN\_Type \*base, uint8\_t mbIdx, const flexcan\_fd\_frame\_t \*pTxFrame)
 

*Writes a FlexCAN FD Message to the Transmit Message Buffer.*
- **status\_t FLEXCAN\_ReadFDRxMb** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_fd\_frame\_t \*pRxFrame)
 

*Reads a FlexCAN FD Message from Receive Message Buffer.*
- **status\_t FLEXCAN\_ReadRxFifo** (CAN\_Type \*base, flexcan\_frame\_t \*pRxFrame)
 

*Reads a FlexCAN Message from Legacy Rx FIFO.*
- **status\_t FLEXCAN\_ReadEnhancedRxFifo** (CAN\_Type \*base, flexcan\_fd\_frame\_t \*pRxFrame)
 

*Reads a FlexCAN Message from Enhanced Rx FIFO.*

## Transactional

- **status\_t FLEXCAN\_TransferFDSendBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_fd\_frame\_t \*pTxFrame)
 

*Performs a polling send transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferFDReceiveBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_fd\_frame\_t \*pRxFrame)
 

*Performs a polling receive transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferFDSendNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*pMbXfer)
 

*Sends a message using IRQ.*
- **status\_t FLEXCAN\_TransferFDReceiveNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*pMbXfer)
 

*Receives a message using IRQ.*
- **void FLEXCAN\_TransferFDAbortSend** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)
 

*Aborts the interrupt driven message send process.*
- **void FLEXCAN\_TransferFDAbortReceive** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)
 

*Aborts the interrupt driven message receive process.*
- **status\_t FLEXCAN\_TransferSendBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_frame\_t \*pTxFrame)
 

*Performs a polling send transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferReceiveBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_frame\_t \*pRxFrame)
 

*Performs a polling receive transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferReceiveFifoBlocking** (CAN\_Type \*base, flexcan\_frame\_t \*pRxFrame)
 

*Performs a polling receive transaction from Legacy Rx FIFO on the CAN bus.*
- **status\_t FLEXCAN\_TransferReceiveEnhancedFifoBlocking** (CAN\_Type \*base, flexcan\_fd\_frame\_t \*pRxFrame)
 

*Performs a polling receive transaction from Enhanced Rx FIFO on the CAN bus.*
- **void FLEXCAN\_TransferCreateHandle** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_callback\_t callback, void \*userData)
 

*Initializes the FlexCAN handle.*
- **status\_t FLEXCAN\_TransferSendNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*pMbXfer)

- Sends a message using IRQ.
- **status\_t FLEXCAN\_TransferReceiveNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*pMbXfer)
  - Receives a message using IRQ.*
- **status\_t FLEXCAN\_TransferReceiveFifoNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_fifo\_transfer\_t \*pFifoXfer)
  - Receives a message from Rx FIFO using IRQ.*
- **status\_t FLEXCAN\_TransferGetReceiveFifoCount** (CAN\_Type \*base, flexcan\_handle\_t \*handle, size\_t \*count)
  - Gets the Legacy Rx Fifo transfer status during a interrupt non-blocking receive.*
- **status\_t FLEXCAN\_TransferReceiveEnhancedFifoNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_fifo\_transfer\_t \*pFifoXfer)
  - Receives a message from Enhanced Rx FIFO using IRQ.*
- static **status\_t FLEXCAN\_TransferGetReceiveEnhancedFifoCount** (CAN\_Type \*base, flexcan\_handle\_t \*handle, size\_t \*count)
  - Gets the Enhanced Rx Fifo transfer status during a interrupt non-blocking receive.*
- **uint32\_t FLEXCAN\_GetTimeStamp** (flexcan\_handle\_t \*handle, uint8\_t mbIdx)
  - Gets the detail index of Mailbox's Timestamp by handle.*
- **void FLEXCAN\_TransferAbortSend** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)
  - Aborts the interrupt driven message send process.*
- **void FLEXCAN\_TransferAbortReceive** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)
  - Aborts the interrupt driven message receive process.*
- **void FLEXCAN\_TransferAbortReceiveFifo** (CAN\_Type \*base, flexcan\_handle\_t \*handle)
  - Aborts the interrupt driven message receive from Rx FIFO process.*
- **void FLEXCAN\_TransferAbortReceiveEnhancedFifo** (CAN\_Type \*base, flexcan\_handle\_t \*handle)
  - Aborts the interrupt driven message receive from Enhanced Rx FIFO process.*
- **void FLEXCAN\_TransferHandleIRQ** (CAN\_Type \*base, flexcan\_handle\_t \*handle)
  - FlexCAN IRQ handle function.*

### 16.2.3 Data Structure Documentation

#### 16.2.3.1 struct flexcan\_memory\_error\_report\_status\_t

This structure contains the memory access properties that caused a memory error access. It is used as the parameter of **FLEXCAN\_GetMemoryErrorReportStatus()** function. And user can use **FLEXCAN\_GetMemoryErrorReportStatus** to get the status of the last memory error access.

#### Data Fields

- **flexcan\_memory\_error\_type\_t errorType**
  - The type of memory error that giving rise to the report.*
- **flexcan\_memory\_access\_type\_t accessType**
  - The type of memory access that giving rise to the memory error.*
- **uint16\_t accessAddress**
  - The address where memory error detected.*

- `uint32_t errorData`  
*The raw data word read from memory with error.*
- `bool byteIsRead`  
*The byte n (0~3) was read or not.*

## Field Documentation

- (1) `flexcan_memory_error_type_t flexcan_memory_error_report_status_t::errorType`
- (2) `flexcan_memory_access_type_t flexcan_memory_error_report_status_t::accessType`
- (3) `uint16_t flexcan_memory_error_report_status_t::accessAddress`
- (4) `uint32_t flexcan_memory_error_report_status_t::errorData`
- (5) `bool flexcan_memory_error_report_status_t::byteIsRead`

The type of error and which bit in byte (n) is affected by the error.

### 16.2.3.2 struct flexcan\_frame\_t

#### Field Documentation

- (1) `uint32_t flexcan_frame_t::timestamp`
- (2) `uint32_t flexcan_frame_t::length`
- (3) `uint32_t flexcan_frame_t::type`
- (4) `uint32_t flexcan_frame_t::format`
- (5) `uint32_t flexcan_frame_t::__pad0__`
- (6) `uint32_t flexcan_frame_t::idhit`
- (7) `uint32_t flexcan_frame_t::id`
- (8) `uint32_t flexcan_frame_t::dataWord0`
- (9) `uint32_t flexcan_frame_t::dataWord1`
- (10) `uint8_t flexcan_frame_t::dataByte3`
- (11) `uint8_t flexcan_frame_t::dataByte2`
- (12) `uint8_t flexcan_frame_t::dataByte1`
- (13) `uint8_t flexcan_frame_t::dataByte0`
- (14) `uint8_t flexcan_frame_t::dataByte7`
- (15) `uint8_t flexcan_frame_t::dataByte6`
- (16) `uint8_t flexcan_frame_t::dataByte5`
- (17) `uint8_t flexcan_frame_t::dataByte4`

### 16.2.3.3 struct flexcan\_fd\_frame\_t

The CAN FD message supporting up to sixty four bytes can be used for a data frame, depending on the length selected for the message buffers. The length should be a enumeration member, see [\\_flexcan\\_fd\\_frame\\_length](#).

#### Data Fields

- `uint32_t idhit`  
*CAN Enhanced Rx FIFO filter hit id (This value is only used in Enhanced Rx FIFO receive mode).*
- `uint32_t timestamp: 16`  
*FlexCAN internal Free-Running Counter Time Stamp.*

- **uint32\_t length:** 4  
*CAN FD frame data length code (DLC), range see [flexcan\\_fd\\_frame\\_length](#), When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value.*
- **uint32\_t type:** 1  
*CAN Frame Type(DATA or REMOTE).*
- **uint32\_t format:** 1  
*CAN Frame Identifier(STD or EXT format).*
- **uint32\_t srr:** 1  
*Substitute Remote request.*
- **uint32\_t \_\_pad0\_\_:** 6  
*Reserved.*
- **uint32\_t esi:** 1  
*Error State Indicator.*
- **uint32\_t brs:** 1  
*Bit Rate Switch.*
- **uint32\_t edl:** 1  
*Extended Data Length.*
- **uint32\_t id:** 29  
*CAN Frame Identifier, should be set using [FLEXCAN\\_ID\\_EXT\(\)](#) or [FLEXCAN\\_ID\\_STD\(\)](#) macro.*
- **uint32\_t dataWord [16]**  
*CAN FD Frame payload, 16 double word maximum.*
- **uint8\_t dataByte3**  
*CAN Frame payload byte3.*
- **uint8\_t dataByte2**  
*CAN Frame payload byte2.*
- **uint8\_t dataByte1**  
*CAN Frame payload byte1.*
- **uint8\_t dataByte0**  
*CAN Frame payload byte0.*
- **uint8\_t dataByte7**  
*CAN Frame payload byte7.*
- **uint8\_t dataByte6**  
*CAN Frame payload byte6.*
- **uint8\_t dataByte5**  
*CAN Frame payload byte5.*
- **uint8\_t dataByte4**  
*CAN Frame payload byte4.*

## Field Documentation

- (1) **uint32\_t flexcan\_fd\_frame\_t::timestamp**
- (2) **uint32\_t flexcan\_fd\_frame\_t::length**

user can use [DLC\\_LENGTH\\_DECODE\(length\)](#) macro to get the number of valid data bytes.

- (3) `uint32_t flexcan_fd_frame_t::type`
- (4) `uint32_t flexcan_fd_frame_t::format`
- (5) `uint32_t flexcan_fd_frame_t::srr`
- (6) `uint32_t flexcan_fd_frame_t::__pad0__`
- (7) `uint32_t flexcan_fd_frame_t::esi`
- (8) `uint32_t flexcan_fd_frame_t::brs`
- (9) `uint32_t flexcan_fd_frame_t::edl`
- (10) `uint32_t flexcan_fd_frame_t::id`
- (11) `uint32_t flexcan_fd_frame_t::dataWord[16]`
- (12) `uint8_t flexcan_fd_frame_t::dataByte3`
- (13) `uint8_t flexcan_fd_frame_t::dataByte2`
- (14) `uint8_t flexcan_fd_frame_t::dataByte1`
- (15) `uint8_t flexcan_fd_frame_t::dataByte0`
- (16) `uint8_t flexcan_fd_frame_t::dataByte7`
- (17) `uint8_t flexcan_fd_frame_t::dataByte6`
- (18) `uint8_t flexcan_fd_frame_t::dataByte5`
- (19) `uint8_t flexcan_fd_frame_t::dataByte4`
- (20) `uint32_t flexcan_fd_frame_t::idhit`

Note

ID HIT offset is changed dynamically according to data length code (DLC), when DLC is 15, they will be located below. Using FLEXCAN\_FixEnhancedRxFifoFrameIdHit API is recommended to ensure this idhit value is correct.

#### 16.2.3.4 struct flexcan\_timing\_config\_t

##### Data Fields

- `uint16_t preDivider`  
*Classic CAN or CAN FD nominal phase bit rate prescaler.*
- `uint8_t rJumpwidth`  
*Classic CAN or CAN FD nominal phase Re-sync Jump Width.*

- `uint8_t phaseSeg1`  
*Classic CAN or CAN FD nominal phase Segment 1.*
- `uint8_t phaseSeg2`  
*Classic CAN or CAN FD nominal phase Segment 2.*
- `uint8_t propSeg`  
*Classic CAN or CAN FD nominal phase Propagation Segment.*
- `uint16_t fpreDivider`  
*CAN FD data phase bit rate prescaler.*
- `uint8_t frJumpwidth`  
*CAN FD data phase Re-sync Jump Width.*
- `uint8_t fphaseSeg1`  
*CAN FD data phase Phase Segment 1.*
- `uint8_t fphaseSeg2`  
*CAN FD data phase Phase Segment 2.*
- `uint8_t fpropSeg`  
*CAN FD data phase Propagation Segment.*

## Field Documentation

- (1) `uint16_t flexcan_timing_config_t::preDivider`
- (2) `uint8_t flexcan_timing_config_t::rJumpwidth`
- (3) `uint8_t flexcan_timing_config_t::phaseSeg1`
- (4) `uint8_t flexcan_timing_config_t::phaseSeg2`
- (5) `uint8_t flexcan_timing_config_t::propSeg`
- (6) `uint16_t flexcan_timing_config_t::fpreDivider`
- (7) `uint8_t flexcan_timing_config_t::frJumpwidth`
- (8) `uint8_t flexcan_timing_config_t::fphaseSeg1`
- (9) `uint8_t flexcan_timing_config_t::fphaseSeg2`
- (10) `uint8_t flexcan_timing_config_t::fpropSeg`

### 16.2.3.5 struct flexcan\_config\_t

**Deprecated** Do not use the baudRate. It has been superceded bitRate

Do not use the baudRateFD. It has been superceded bitRateFD

## Data Fields

- `flexcan_clock_source_t clkSrc`  
*Clock source for FlexCAN Protocol Engine.*
- `flexcan_wake_up_source_t wakeupSrc`  
*Wake up source selection.*

- `uint8_t maxMbNum`  
*The maximum number of Message Buffers used by user.*
- `bool enableLoopBack`  
*Enable or Disable Loop Back Self Test Mode.*
- `bool enableTimerSync`  
*Enable or Disable Timer Synchronization.*
- `bool enableSelfWakeup`  
*Enable or Disable Self Wakeup Mode.*
- `bool enableIndividMask`  
*Enable or Disable Rx Individual Mask and Queue feature.*
- `bool disableSelfReflection`  
*Enable or Disable Self Reflection.*
- `bool enableListenOnlyMode`  
*Enable or Disable Listen Only Mode.*
- `bool enableSupervisorMode`  
*Enable or Disable Supervisor Mode, enable this mode will make registers allow only Supervisor access.*
- `bool enableDoze`  
*Enable or Disable Doze Mode.*
- `bool enableMemoryErrorControl`  
*Enable or Disable the memory errors detection and correction mechanism.*
- `bool enableNonCorrectableErrorEnterFreeze`  
*Enable or Disable Non-Correctable Errors In FlexCAN Access Put Device In Freeze Mode.*
- `bool enableTransceiverDelayMeasure`  
*Enable or Disable the transceiver delay measurement, when it is enabled, then the secondary sample point position is determined by the sum of the transceiver delay measurement plus the enhanced TDC offset.*
- `uint32_t baudRate`  
*FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.*
- `uint32_t baudRateFD`  
*FlexCAN FD bit rate in bps, for CANFD data phase.*
- `uint32_t bitRate`  
*FlexCAN bit rate in bps, for classical CAN or CANFD nominal phase.*
- `uint32_t bitRateFD`  
*FlexCAN FD bit rate in bps, for CANFD data phase.*

## Field Documentation

- (1) `uint32_t flexcan_config_t::baudRate`
- (2) `uint32_t flexcan_config_t::baudRateFD`
- (3) `uint32_t flexcan_config_t::bitRate`
- (4) `uint32_t flexcan_config_t::bitRateFD`
- (5) `flexcan_clock_source_t flexcan_config_t::clkSrc`
- (6) `flexcan_wake_up_source_t flexcan_config_t::wakeupSrc`
- (7) `uint8_t flexcan_config_t::maxMbNum`
- (8) `bool flexcan_config_t::enableLoopBack`
- (9) `bool flexcan_config_t::enableTimerSync`
- (10) `bool flexcan_config_t::enableSelfWakeup`
- (11) `bool flexcan_config_t::enableIndividMask`
- (12) `bool flexcan_config_t::disableSelfReception`
- (13) `bool flexcan_config_t::enableListenOnlyMode`
- (14) `bool flexcan_config_t::enableSupervisorMode`
- (15) `bool flexcan_config_t::enableDoze`
- (16) `bool flexcan_config_t::enableMemoryErrorControl`
- (17) `bool flexcan_config_t::enableNonCorrectableErrorEnterFreeze`
- (18) `bool flexcan_config_t::enableTransceiverDelayMeasure`

### 16.2.3.6 struct flexcan\_rx\_mb\_config\_t

This structure is used as the parameter of [FLEXCAN\\_SetRxMbConfig\(\)](#) function. The [FLEXCAN\\_SetRxMbConfig\(\)](#) function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

## Data Fields

- `uint32_t id`  
*CAN Message Buffer Frame Identifier, should be set using [FLEXCAN\\_ID\\_EXT\(\)](#) or [FLEXCAN\\_ID\\_STD\(\)](#) macro.*

- **flexcan\_frame\_format\_t format**  
*CAN Frame Identifier format(Standard or Extend).*
- **flexcan\_frame\_type\_t type**  
*CAN Frame Type(Data or Remote).*

**Field Documentation**

- (1) **uint32\_t flexcan\_rx\_mb\_config\_t::id**
- (2) **flexcan\_frame\_format\_t flexcan\_rx\_mb\_config\_t::format**
- (3) **flexcan\_frame\_type\_t flexcan\_rx\_mb\_config\_t::type**

**16.2.3.7 struct flexcan\_rx\_fifo\_config\_t****Data Fields**

- **uint32\_t \* idFilterTable**  
*Pointer to the FlexCAN Legacy Rx FIFO identifier filter table.*
- **uint8\_t idFilterNum**  
*The FlexCAN Legacy Rx FIFO Filter elements quantity.*
- **flexcan\_rx\_fifo\_filter\_type\_t idFilterType**  
*The FlexCAN Legacy Rx FIFO Filter type.*
- **flexcan\_rx\_fifo\_priority\_t priority**  
*The FlexCAN Legacy Rx FIFO receive priority.*

**Field Documentation**

- (1) **uint32\_t\* flexcan\_rx\_fifo\_config\_t::idFilterTable**
- (2) **uint8\_t flexcan\_rx\_fifo\_config\_t::idFilterNum**
- (3) **flexcan\_rx\_fifo\_filter\_type\_t flexcan\_rx\_fifo\_config\_t::idFilterType**
- (4) **flexcan\_rx\_fifo\_priority\_t flexcan\_rx\_fifo\_config\_t::priority**

**16.2.3.8 struct flexcan\_enhanced\_rx\_fifo\_std\_id\_filter\_t****Data Fields**

- **uint32\_t filterType: 2**  
*FlexCAN internal Free-Running Counter Time Stamp.*
- **uint32\_t rtr1: 1**  
*CAN FD frame data length code (DLC), range see [flexcan\\_fd\\_frame\\_length](#), When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value.*
- **uint32\_t std1: 11**  
*CAN Frame Type(DATA or REMOTE).*
- **uint32\_t rtr2: 1**  
*CAN Frame Identifier(STD or EXT format).*
- **uint32\_t std2: 11**  
*Substitute Remote request.*

## Field Documentation

- (1) `uint32_t flexcan_enhanced_rx_fifo_std_id_filter_t::filterType`
- (2) `uint32_t flexcan_enhanced_rx_fifo_std_id_filter_t::rtr1`

user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

- (3) `uint32_t flexcan_enhanced_rx_fifo_std_id_filter_t::std1`
- (4) `uint32_t flexcan_enhanced_rx_fifo_std_id_filter_t::rtr2`
- (5) `uint32_t flexcan_enhanced_rx_fifo_std_id_filter_t::std2`

### 16.2.3.9 struct flexcan\_enhanced\_rx\_fifo\_ext\_id\_filter\_t

## Data Fields

- `uint32_t filterType`: 2  
*FlexCAN internal Free-Running Counter Time Stamp.*
- `uint32_t rtr1`: 1  
*CAN FD frame data length code (DLC), range see `flexcan_fd_frame_length`, When the length <= 8, it equal to the data length, otherwise the number of valid frame data is not equal to the length value.*
- `uint32_t std1`: 29  
*CAN Frame Type(DATA or REMOTE).*
- `uint32_t rtr2`: 1  
*CAN Frame Identifier(STD or EXT format).*
- `uint32_t std2`: 29  
*Substitute Remote request.*

## Field Documentation

- (1) `uint32_t flexcan_enhanced_rx_fifo_ext_id_filter_t::filterType`
- (2) `uint32_t flexcan_enhanced_rx_fifo_ext_id_filter_t::rtr1`

user can use `DLC_LENGTH_DECODE(length)` macro to get the number of valid data bytes.

- (3) `uint32_t flexcan_enhanced_rx_fifo_ext_id_filter_t::std1`
- (4) `uint32_t flexcan_enhanced_rx_fifo_ext_id_filter_t::rtr2`
- (5) `uint32_t flexcan_enhanced_rx_fifo_ext_id_filter_t::std2`

### 16.2.3.10 struct flexcan\_enhanced\_rx\_fifo\_config\_t

## Data Fields

- `uint32_t * idFilterTable`  
Pointer to the FlexCAN Enhanced Rx FIFO identifier filter table, each table member occupies 32 bit word, table size should be equal to `idFilterNum`.

- `uint8_t idFilterPairNum`

*idFilterPairNum* is the Enhanced Rx FIFO identifier filter element pair numbers, each pair of filter elements occupies 2 words and can consist of one extended ID filter element or two standard ID filter elements.

- `uint8_t extendIdFilterNum`

The number of extended ID filter element items in the FlexCAN enhanced Rx FIFO identifier filter table, each extended-ID filter element occupies 2 words, `extendIdFilterNum` need less than or equal to `idFilterPairNum`.

- `uint8_t fifoWatermark`

*(fifoWatermark + 1)* is the minimum number of CAN messages stored in the Enhanced RX FIFO which can trigger FIFO watermark interrupt or a DMA request.

- `flexcan_efifo_dma_per_read_length_t dmaPerReadLength`

Define the length of each read of the Enhanced RX FIFO element by the DAM, see `_flexcan_fd_frame_length`.

- `flexcan_rx_fifo_priority_t priority`

The FlexCAN Enhanced Rx FIFO receive priority.

## Field Documentation

(1) `uint32_t* flexcan_enhanced_rx_fifo_config_t::idFilterTable`

There are two types of Enhanced Rx FIFO filter elements that can be stored in table : extended-ID filter element (1 word, occupies 1 table members) and standard-ID filter element (2 words, occupies 2 table members), the extended-ID filter element needs to be placed in front of the table.

(2) `uint8_t flexcan_enhanced_rx_fifo_config_t::idFilterPairNum`

(3) `uint8_t flexcan_enhanced_rx_fifo_config_t::extendIdFilterNum`

(4) `uint8_t flexcan_enhanced_rx_fifo_config_t::fifoWatermark`

(5) `flexcan_efifo_dma_per_read_length_t flexcan_enhanced_rx_fifo_config_t::dmaPerReadLength`

(6) `flexcan_rx_fifo_priority_t flexcan_enhanced_rx_fifo_config_t::priority`

### 16.2.3.11 struct flexcan\_mb\_transfer\_t

#### Data Fields

- `flexcan_frame_t * frame`

*The buffer of CAN Message to be transfer.*

- `uint8_t mblIdx`

*The index of Message buffer used to transfer Message.*

**Field Documentation**

- (1) **flexcan\_frame\_t\* flexcan\_mb\_transfer\_t::frame**
- (2) **uint8\_t flexcan\_mb\_transfer\_t::mbIdx**

**16.2.3.12 struct flexcan\_fifo\_transfer\_t****Data Fields**

- **flexcan\_fd\_frame\_t \* framefd**  
*The buffer of CAN Message to be received from Enhanced Rx FIFO.*
- **flexcan\_frame\_t \* frame**  
*The buffer of CAN Message to be received from Legacy Rx FIFO.*
- **size\_t frameNum**  
*Number of CAN Message need to be received from Legacy or Enhanced Rx FIFO.*

**Field Documentation**

- (1) **flexcan\_fd\_frame\_t\* flexcan\_fifo\_transfer\_t::framefd**
- (2) **flexcan\_frame\_t\* flexcan\_fifo\_transfer\_t::frame**
- (3) **size\_t flexcan\_fifo\_transfer\_t::frameNum**

**16.2.3.13 struct \_flexcan\_handle**

FlexCAN handle structure definition.

**Data Fields**

- **flexcan\_transfer\_callback\_t callback**  
*Callback function.*
- **void \* userData**  
*FlexCAN callback function parameter.*
- **flexcan\_frame\_t \*volatile mbFrameBuf [CAN\_WORD1\_COUNT]**  
*The buffer for received CAN data from Message Buffers.*
- **flexcan\_fd\_frame\_t \*volatile mbFDFrameBuf [CAN\_WORD1\_COUNT]**  
*The buffer for received CAN FD data from Message Buffers.*
- **flexcan\_frame\_t \*volatile rxFifoFrameBuf**  
*The buffer for received CAN data from Legacy Rx FIFO.*
- **flexcan\_fd\_frame\_t \*volatile rxFifoFDFrameBuf**  
*The buffer for received CAN FD data from Enhanced Rx FIFO.*
- **size\_t rxFifoFrameNum**  
*The number of CAN messages remaining to be received from Legacy or Enhanced Rx FIFO.*
- **size\_t rxFifoTransferTotalNum**  
*Total CAN Message number need to be received from Legacy or Enhanced Rx FIFO.*
- **volatile uint8\_t mbState [CAN\_WORD1\_COUNT]**  
*Message Buffer transfer state.*
- **volatile uint8\_t rxFifoState**

- volatile uint32\_t **timestamp** [CAN\_WORD1\_COUNT]  
*Mailbox transfer timestamp.*

### Field Documentation

- (1) **flexcan\_transfer\_callback\_t** **flexcan\_handle\_t::callback**
- (2) **void\*** **flexcan\_handle\_t::userData**
- (3) **flexcan\_frame\_t\*** **volatile flexcan\_handle\_t::mbFrameBuf**[CAN\_WORD1\_COUNT]
- (4) **flexcan\_fd\_frame\_t\*** **volatile flexcan\_handle\_t::mbFDFrameBuf**[CAN\_WORD1\_COUNT]
- (5) **flexcan\_frame\_t\*** **volatile flexcan\_handle\_t::rxFifoFrameBuf**
- (6) **flexcan\_fd\_frame\_t\*** **volatile flexcan\_handle\_t::rxFifoFDFrameBuf**
- (7) **size\_t** **flexcan\_handle\_t::rxFifoFrameNum**
- (8) **size\_t** **flexcan\_handle\_t::rxFifoTransferTotalNum**
- (9) **volatile uint8\_t** **flexcan\_handle\_t::mbState**[CAN\_WORD1\_COUNT]
- (10) **volatile uint8\_t** **flexcan\_handle\_t::rxFifoState**
- (11) **volatile uint32\_t** **flexcan\_handle\_t::timestamp**[CAN\_WORD1\_COUNT]

### 16.2.4 Macro Definition Documentation

**16.2.4.1 #define FSL\_FLEXCAN\_DRIVER\_VERSION (MAKE\_VERSION(2, 11, 4))**

**16.2.4.2 #define DLC\_LENGTH\_DECODE( *dlic* ) (((dlic) <= 8U) ? (dlic) : (((dlic) <= 12U) ? ((dlic)-6U) \* 4U) : (((dlic)-11U) \* 16U)))**

**16.2.4.3 #define FLEXCAN\_ID\_STD( *id* ) (((uint32\_t)((uint32\_t)(*id*) << CAN\_ID\_STD\_SHIFT)) & CAN\_ID\_STD\_MASK)**

Standard Frame ID helper macro.

**16.2.4.4 #define FLEXCAN\_ID\_EXT( *id* )**

**Value:**

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

**16.2.4.5 #define FLEXCAN\_RX\_MB\_STD\_MASK( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_STD(id))
```

Standard Rx Message Buffer Mask helper macro.

**16.2.4.6 #define FLEXCAN\_RX\_MB\_EXT\_MASK( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_EXT(id))
```

**16.2.4.7 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(FLEXCAN_ID_STD(id) << 1))
```

Standard Rx FIFO Mask helper macro Type A helper macro.

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

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(((uint32_t)(id)&0x7FF) << 19))
```

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

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
(((uint32_t)(id)&0x7FF) << 3))
```

- 16.2.4.10 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH( *id* ) (((uint32\_t)(*id*)&0x7F8) << 21)**
- 16.2.4.11 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH( *id* ) (((uint32\_t)(*id*)&0x7F8) << 13)**
- 16.2.4.12 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW( *id* ) (((uint32\_t)(*id*)&0x7F8) << 5)**
- 16.2.4.13 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW( *id* ) (((uint32\_t)(*id*)&0x7F8) >> 3)**
- 16.2.4.14 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )**

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
 (FLEXCAN_ID_EXT(id) << 1))
```

- 16.2.4.15 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
(\
 ((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
 ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) \
 << 1)) \ \
```

- 16.2.4.16 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )**

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
 ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >>
 15)) \
```

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

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

**Value:**

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

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

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

**16.2.4.20 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW( *id* ) ((FLEXCAN\_ID\_EXT(*id*) & 0x1FE00000) >> 21)****16.2.4.21 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* ) FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A(*id*, *rtr*, *ide*)**

Standard Rx FIFO Filter helper macro Type A helper macro.

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

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

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

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

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

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(id) \
```

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

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(id) \
```

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

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(\
 id)
```

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

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(\
 id)
```

**16.2.4.28 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A(*id*, *rtr*, *ide*)****16.2.4.29 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )****Value:**

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

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

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

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

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(\
 id)
```

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

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(\
 _id)
```

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

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(\
 _id)
```

**16.2.4.34 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_LOW( *id* ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW(*id*)****16.2.4.35 #define ENHANCED\_RX\_FIFO\_FSCH( *x* ) (((uint32\_t)((uint32\_t)(*x*)) << 30)) & 0xC0000000U)****16.2.4.36 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_STD\_MASK\_AND\_FILTER( *id*, *rtr*, *id\_mask*, *rtr\_mask* )(ENHANCED\_RX\_FIFO\_FSCH(0x0) | RTR\_STD\_HIGH(*rtr*) | ID\_STD\_HIGH(*id*) | RTR\_STD\_LOW(*rtr\_mask*) | ID\_STD\_LOW(*id\_mask*))****16.2.4.37 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_STD\_FILTER\_WITH\_RANGE( *id\_upper*, *rtr*, *id\_lower*, *rtr\_mask* )****Value:**

```
(ENHANCED_RX_FIFO_FSCH(0x1) | RTR_STD_HIGH(rtr) | ID_STD_HIGH(id_upper) | RTR_STD_LOW(
 rtr_mask) | \
 ID_STD_LOW(id_lower))
```

- 16.2.4.38 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_STD\_TWO\_FILTERS( *id1*, *rtr1*,  
*id2*, *rtr2* ) (ENHANCED\_RX\_FIFO\_FSCH(0x2) | RTR\_STD\_HIGH(*rtr1*) |  
ID\_STD\_HIGH(*id1*) | RTR\_STD\_LOW(*rtr2*) | ID\_STD\_LOW(*id2*))
- 16.2.4.39 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_MASK\_AND\_FILTER\_LOW( *id*,  
*rtr* ) (ENHANCED\_RX\_FIFO\_FSCH(0x0) | RTR\_EXT(*rtr*) | ID\_EXT(*id*))
- 16.2.4.40 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_MASK\_AND\_FILTER\_HIGH( *id\_mask*,  
*rtr\_mask* ) (ENHANCED\_RX\_FIFO\_FSCH(0x0) |  
RTR\_EXT(*rtr\_mask*) | ID\_EXT(*id\_mask*))
- 16.2.4.41 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_FILTER\_WITH\_RANGE\_LOW( *id\_upper*, *rtr* ) (ENHANCED\_RX\_FIFO\_FSCH(0x1) | RTR\_EXT(*rtr*) |  
ID\_EXT(*id\_upper*))
- 16.2.4.42 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_FILTER\_WITH\_RANGE\_HIGH( *id\_lower*, *rtr\_mask* ) (ENHANCED\_RX\_FIFO\_FSCH(0x1) |  
RTR\_EXT(*rtr\_mask*) | ID\_EXT(*id\_lower*))
- 16.2.4.43 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_TWO\_FILTERS\_LOW( *id2*, *rtr2* ) (ENHANCED\_RX\_FIFO\_FSCH(0x2) | RTR\_EXT(*rtr2*) | ID\_EXT(*id2*))
- 16.2.4.44 #define FLEXCAN\_ENHANCED\_RX\_FIFO\_EXT\_TWO\_FILTERS\_HIGH( *id1*, *rtr1* ) (ENHANCED\_RX\_FIFO\_FSCH(0x2) | RTR\_EXT(*rtr1*) | ID\_EXT(*id1*))
- 16.2.4.45 #define FLEXCAN\_EFIFO\_INT\_MASK( *x* ) (((uint64\_t)((uint64\_t)(*x*) << 32)) &  
0xF00000000000000U)
- 16.2.4.46 #define E\_RX\_FIFO( *base* ) ((uintptr\_t)(*base*) + 0x2000U)
- 16.2.4.47 #define FLEXCAN\_CALLBACK( *x* ) void(*x*)(CAN\_Type \* *base*, flexcan\_handle\_t  
\* *handle*, status\_t *status*, uint64\_t *result*, void \* *userData*)

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus\_FLEXCAN\_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

## 16.2.5 Enumeration Type Documentation

### 16.2.5.1 anonymous enum

Enumerator

*kStatus\_FLEXCAN\_TxBusy* Tx Message Buffer is Busy.  
*kStatus\_FLEXCAN\_TxIdle* Tx Message Buffer is Idle.  
*kStatus\_FLEXCAN\_TxSwitchToRx* Remote Message is send out and Message buffer changed to Receive one.  
*kStatus\_FLEXCAN\_RxBusy* Rx Message Buffer is Busy.  
*kStatus\_FLEXCAN\_RxIdle* Rx Message Buffer is Idle.  
*kStatus\_FLEXCAN\_RxOverflow* Rx Message Buffer is Overflowed.  
*kStatus\_FLEXCAN\_RxFifoBusy* Rx Message FIFO is Busy.  
*kStatus\_FLEXCAN\_RxFifoIdle* Rx Message FIFO is Idle.  
*kStatus\_FLEXCAN\_RxFifoOverflow* Rx Message FIFO is overflowed.  
*kStatus\_FLEXCAN\_RxFifoWarning* Rx Message FIFO is almost overflowed.  
*kStatus\_FLEXCAN\_RxFifoDisabled* Rx Message FIFO is disabled during reading.  
*kStatus\_FLEXCAN\_ErrorStatus* FlexCAN Module Error and Status.  
*kStatus\_FLEXCAN\_WakeUp* FlexCAN is waken up from STOP mode.  
*kStatus\_FLEXCAN\_UnHandled* UnHadled Interrupt asserted.  
*kStatus\_FLEXCAN\_RxRemote* Rx Remote Message Received in Mail box.  
*kStatus\_FLEXCAN\_RxFifoUnderflow* Enhanced Rx Message FIFO is underflow.

### 16.2.5.2 enum flexcan\_frame\_format\_t

Enumerator

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

### 16.2.5.3 enum flexcan\_frame\_type\_t

Enumerator

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

### 16.2.5.4 enum flexcan\_clock\_source\_t

**Deprecated** Do not use the kFLEXCAN\_ClkSrcOs. It has been superceded kFLEXCAN\_ClkSrc0  
Do not use the kFLEXCAN\_ClkSrcPeri. It has been superceded kFLEXCAN\_ClkSrc1

Enumerator

***kFLEXCAN\_ClkSrcOsc*** FlexCAN Protocol Engine clock from Oscillator.

***kFLEXCAN\_ClkSrcPeri*** FlexCAN Protocol Engine clock from Peripheral Clock.

***kFLEXCAN\_ClkSrc0*** FlexCAN Protocol Engine clock selected by user as SRC == 0.

***kFLEXCAN\_ClkSrc1*** FlexCAN Protocol Engine clock selected by user as SRC == 1.

#### 16.2.5.5 enum flexcan\_wake\_up\_source\_t

Enumerator

***kFLEXCAN\_WakeupSrcUnfiltered*** FlexCAN uses unfiltered Rx input to detect edge.

***kFLEXCAN\_WakeupSrcFiltered*** FlexCAN uses filtered Rx input to detect edge.

#### 16.2.5.6 enum flexcan\_rx\_fifo\_filter\_type\_t

Enumerator

***kFLEXCAN\_RxFifoFilterTypeA*** One full ID (standard and extended) per ID Filter element.

***kFLEXCAN\_RxFifoFilterTypeB*** Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

***kFLEXCAN\_RxFifoFilterTypeC*** Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

***kFLEXCAN\_RxFifoFilterTypeD*** All frames rejected.

#### 16.2.5.7 enum flexcan\_mb\_size\_t

Enumerator

***kFLEXCAN\_8BperMB*** Selects 8 bytes per Message Buffer.

***kFLEXCAN\_16BperMB*** Selects 16 bytes per Message Buffer.

***kFLEXCAN\_32BperMB*** Selects 32 bytes per Message Buffer.

***kFLEXCAN\_64BperMB*** Selects 64 bytes per Message Buffer.

#### 16.2.5.8 enum \_flexcan\_fd\_frame\_length

For Tx, when the Data size corresponding to DLC value stored in the MB selected for transmission is larger than the MB Payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC. For Rx, when the Data size corresponding to DLC value received from the CAN bus is larger than the MB Payload size, the high order bytes that do not fit the Payload size will lose.

Enumerator

***kFLEXCAN\_0BperFrame*** Frame contains 0 valid data bytes.

|                             |                                     |
|-----------------------------|-------------------------------------|
| <i>kFLEXCAN_1BperFrame</i>  | Frame contains 1 valid data bytes.  |
| <i>kFLEXCAN_2BperFrame</i>  | Frame contains 2 valid data bytes.  |
| <i>kFLEXCAN_3BperFrame</i>  | Frame contains 3 valid data bytes.  |
| <i>kFLEXCAN_4BperFrame</i>  | Frame contains 4 valid data bytes.  |
| <i>kFLEXCAN_5BperFrame</i>  | Frame contains 5 valid data bytes.  |
| <i>kFLEXCAN_6BperFrame</i>  | Frame contains 6 valid data bytes.  |
| <i>kFLEXCAN_7BperFrame</i>  | Frame contains 7 valid data bytes.  |
| <i>kFLEXCAN_8BperFrame</i>  | Frame contains 8 valid data bytes.  |
| <i>kFLEXCAN_12BperFrame</i> | Frame contains 12 valid data bytes. |
| <i>kFLEXCAN_16BperFrame</i> | Frame contains 16 valid data bytes. |
| <i>kFLEXCAN_20BperFrame</i> | Frame contains 20 valid data bytes. |
| <i>kFLEXCAN_24BperFrame</i> | Frame contains 24 valid data bytes. |
| <i>kFLEXCAN_32BperFrame</i> | Frame contains 32 valid data bytes. |
| <i>kFLEXCAN_48BperFrame</i> | Frame contains 48 valid data bytes. |
| <i>kFLEXCAN_64BperFrame</i> | Frame contains 64 valid data bytes. |

### 16.2.5.9 enum flexcan\_efifo\_dma\_per\_read\_length\_t

Enumerator

|                               |                                                              |
|-------------------------------|--------------------------------------------------------------|
| <i>kFLEXCAN_1WordPerRead</i>  | Transfer 1 32-bit words (CS).                                |
| <i>kFLEXCAN_2WordPerRead</i>  | Transfer 2 32-bit words (CS + ID).                           |
| <i>kFLEXCAN_3WordPerRead</i>  | Transfer 3 32-bit words (CS + ID + 1~4 bytes data).          |
| <i>kFLEXCAN_4WordPerRead</i>  | Transfer 4 32-bit words (CS + ID + 5~8 bytes data).          |
| <i>kFLEXCAN_5WordPerRead</i>  | Transfer 5 32-bit words (CS + ID + 9~12 bytes data).         |
| <i>kFLEXCAN_6WordPerRead</i>  | Transfer 6 32-bit words (CS + ID + 13~16 bytes data).        |
| <i>kFLEXCAN_7WordPerRead</i>  | Transfer 7 32-bit words (CS + ID + 17~20 bytes data).        |
| <i>kFLEXCAN_8WordPerRead</i>  | Transfer 8 32-bit words (CS + ID + 21~24 bytes data).        |
| <i>kFLEXCAN_9WordPerRead</i>  | Transfer 9 32-bit words (CS + ID + 25~28 bytes data).        |
| <i>kFLEXCAN_10WordPerRead</i> | Transfer 10 32-bit words (CS + ID + 29~32 bytes data).       |
| <i>kFLEXCAN_11WordPerRead</i> | Transfer 11 32-bit words (CS + ID + 33~36 bytes data).       |
| <i>kFLEXCAN_12WordPerRead</i> | Transfer 12 32-bit words (CS + ID + 37~40 bytes data).       |
| <i>kFLEXCAN_13WordPerRead</i> | Transfer 13 32-bit words (CS + ID + 41~44 bytes data).       |
| <i>kFLEXCAN_14WordPerRead</i> | Transfer 14 32-bit words (CS + ID + 45~48 bytes data).       |
| <i>kFLEXCAN_15WordPerRead</i> | Transfer 15 32-bit words (CS + ID + 49~52 bytes data).       |
| <i>kFLEXCAN_16WordPerRead</i> | Transfer 16 32-bit words (CS + ID + 53~56 bytes data).       |
| <i>kFLEXCAN_17WordPerRead</i> | Transfer 17 32-bit words (CS + ID + 57~60 bytes data).       |
| <i>kFLEXCAN_18WordPerRead</i> | Transfer 18 32-bit words (CS + ID + 61~64 bytes data).       |
| <i>kFLEXCAN_19WordPerRead</i> | Transfer 19 32-bit words (CS + ID + 64 bytes data + ID HIT). |

### 16.2.5.10 enum flexcan\_rx\_fifo\_priority\_t

The matching process starts from the Rx MB(or Enhanced/Legacy Rx FIFO) with higher priority. If no MB(or Enhanced/Legacy Rx FIFO filter) is satisfied, the matching process goes on with the Enhanced/-

Legacy Rx FIFO(or Rx MB) with lower priority.

Enumerator

***kFLEXCAN\_RxFifoPrioLow*** Matching process start from Rx Message Buffer first.

***kFLEXCAN\_RxFifoPrioHigh*** Matching process start from Enhanced/Legacy Rx FIFO first.

### 16.2.5.11 enum \_flexcan\_interrupt\_enable

This provides constants for the FlexCAN interrupt enable enumerations for use in the FlexCAN functions.

Note

FlexCAN Message Buffers and Legacy Rx FIFO interrupts not included in.

Enumerator

***kFLEXCAN\_BusOffInterruptEnable*** Bus Off interrupt, use bit 15.

***kFLEXCAN\_ErrorInterruptEnable*** CAN Error interrupt, use bit 14.

***kFLEXCAN\_TxWarningInterruptEnable*** Tx Warning interrupt, use bit 11.

***kFLEXCAN\_RxWarningInterruptEnable*** Rx Warning interrupt, use bit 10.

***kFLEXCAN\_WakeUpInterruptEnable*** Self Wake Up interrupt, use bit 26.

***kFLEXCAN\_FDErrorInterruptEnable*** CAN FD Error interrupt, use bit 31. Enhanced Rx FIFO Underflow interrupt, use high word bit 31.

***kFLEXCAN\_ERxFifoUnderflowInterruptEnable*** Enhanced Rx FIFO Overflow interrupt, use high word bit 30.

***kFLEXCAN\_ERxFifoOverflowInterruptEnable*** Enhanced Rx FIFO Watermark interrupt, use high word bit 29.

***kFLEXCAN\_ERxFifoWatermarkInterruptEnable*** Enhanced Rx FIFO Data Available interrupt, use high word bit 28.

***kFLEXCAN\_HostAccessNCErrorInterruptEnable*** Host Access With Non-Correctable Errors interrupt, use high word bit 0.

***kFLEXCAN\_FlexCanAccessNCErrorInterruptEnable*** FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.

***kFLEXCAN\_HostOrFlexCanCErrorInterruptEnable*** Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

### 16.2.5.12 enum \_flexcan\_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions.

## Note

The CPU read action clears the bits corresponding to the FLEXCAN\_ErrorFlag macro, therefore user need to read status flags and distinguish which error is occur using [\\_flexcan\\_error\\_flags](#) enumerations.

## Enumerator

***kFLEXCAN\_ErrorOverrunFlag*** Error Overrun Status.  
***kFLEXCAN\_FDErrorIntFlag*** CAN FD Error Interrupt Flag.  
***kFLEXCAN\_BusoffDoneIntFlag*** Bus Off process completed Interrupt Flag.  
***kFLEXCAN\_SynchFlag*** CAN Synchronization Status.  
***kFLEXCAN\_TxWarningIntFlag*** Tx Warning Interrupt Flag.  
***kFLEXCAN\_RxWarningIntFlag*** Rx Warning Interrupt Flag.  
***kFLEXCAN\_IdleFlag*** FlexCAN In IDLE Status.  
***kFLEXCAN\_FaultConfinementFlag*** FlexCAN Fault Confinement State.  
***kFLEXCAN\_TransmittingFlag*** FlexCAN In Transmission Status.  
***kFLEXCAN\_ReceivingFlag*** FlexCAN In Reception Status.  
***kFLEXCAN\_BusOffIntFlag*** Bus Off Interrupt Flag.  
***kFLEXCAN\_ErrorIntFlag*** CAN Error Interrupt Flag.  
***kFLEXCAN\_WakeUpIntFlag*** Self Wake-Up Interrupt Flag.  
***kFLEXCAN\_ERxFifoUnderflowIntFlag*** Enhanced Rx FIFO underflow Interrupt Flag.  
***kFLEXCAN\_ERxFifoOverflowIntFlag*** Enhanced Rx FIFO overflow Interrupt Flag.  
***kFLEXCAN\_ERxFifoWatermarkIntFlag*** Enhanced Rx FIFO watermark Interrupt Flag.  
***kFLEXCAN\_ERxFifoDataAvlIntFlag*** Enhanced Rx FIFO data available Interrupt Flag.  
***kFLEXCAN\_ERxFifoEmptyFlag*** Enhanced Rx FIFO empty status.  
***kFLEXCAN\_ERxFifoFullFlag*** Enhanced Rx FIFO full status.  
***kFLEXCAN\_HostAccessNonCorrectableErrorIntFlag*** Host Access With Non-Correctable Error Interrupt Flag.  
***kFLEXCAN\_FlexCanAccessNonCorrectableErrorIntFlag*** FlexCAN Access With Non-Correctable Error Interrupt Flag.  
***kFLEXCAN\_CorrectableErrorIntFlag*** Correctable Error Interrupt Flag.  
***kFLEXCAN\_HostAccessNonCorrectableErrorOverrunFlag*** Host Access With Non-Correctable Error Interrupt Overrun Flag.  
***kFLEXCAN\_FlexCanAccessNonCorrectableErrorOverrunFlag*** FlexCAN Access With Non-Correctable Error Interrupt Overrun Flag.  
***kFLEXCAN\_CorrectableErrorOverrunFlag*** Correctable Error Interrupt Overrun Flag.  
***kFLEXCAN\_AllMemoryErrorFlag*** All Memory Error Flags.

**16.2.5.13 enum \_flexcan\_error\_flags**

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN\_ErrorFlag in [\\_flexcan\\_flags](#) enumerations to determine which error is generated.

Enumerator

- kFLEXCAN\_FDStuffingError*** Stuffing Error.
- kFLEXCAN\_FDFormError*** Form Error.
- kFLEXCAN\_FDCrcError*** Cyclic Redundancy Check Error.
- kFLEXCAN\_FDBit0Error*** Unable to send dominant bit.
- kFLEXCAN\_FDBit1Error*** Unable to send recessive bit.
- kFLEXCAN\_TxErrorWarningFlag*** Tx Error Warning Status.
- kFLEXCAN\_RxErrorWarningFlag*** Rx Error Warning Status.
- kFLEXCAN\_StuffingError*** Stuffing Error.
- kFLEXCAN\_FormError*** Form Error.
- kFLEXCAN\_CrcError*** Cyclic Redundancy Check Error.
- kFLEXCAN\_AckError*** Received no ACK on transmission.
- kFLEXCAN\_Bit0Error*** Unable to send dominant bit.
- kFLEXCAN\_Bit1Error*** Unable to send recessive bit.

#### 16.2.5.14 anonymous enum

The FlexCAN Legacy Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Enumerator

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

#### 16.2.5.15 enum flexcan\_memory\_error\_type\_t

Enumerator

- kFLEXCAN\_CorrectableError*** The memory error is correctable which means one bit error.
- kFLEXCAN\_NonCorrectableError*** The memory error is non-correctable which means two bit errors.

#### 16.2.5.16 enum flexcan\_memory\_access\_type\_t

Enumerator

- kFLEXCAN\_MoveOutFlexCanAccess*** The memory error was detected during move-out FlexCAN access.
- kFLEXCAN\_MoveInAccess*** The memory error was detected during move-in FlexCAN access.
- kFLEXCAN\_TxArbitrationAccess*** The memory error was detected during Tx Arbitration FlexCAN access.

***kFLEXCAN\_RxMatchingAccess*** The memory error was detected during Rx Matching FlexCAN access.

***kFLEXCAN\_MoveOutHostAccess*** The memory error was detected during Rx Matching Host (CPU) access.

### 16.2.5.17 enum flexcan\_byte\_error\_syndrome\_t

Enumerator

***kFLEXCAN\_NoError*** No bit error in this byte.

***kFLEXCAN\_ParityBits0Error*** Parity bit 0 error in this byte.

***kFLEXCAN\_ParityBits1Error*** Parity bit 1 error in this byte.

***kFLEXCAN\_ParityBits2Error*** Parity bit 2 error in this byte.

***kFLEXCAN\_ParityBits3Error*** Parity bit 3 error in this byte.

***kFLEXCAN\_ParityBits4Error*** Parity bit 4 error in this byte.

***kFLEXCAN\_DataBits0Error*** Data bit 0 error in this byte.

***kFLEXCAN\_DataBits1Error*** Data bit 1 error in this byte.

***kFLEXCAN\_DataBits2Error*** Data bit 2 error in this byte.

***kFLEXCAN\_DataBits3Error*** Data bit 3 error in this byte.

***kFLEXCAN\_DataBits4Error*** Data bit 4 error in this byte.

***kFLEXCAN\_DataBits5Error*** Data bit 5 error in this byte.

***kFLEXCAN\_DataBits6Error*** Data bit 6 error in this byte.

***kFLEXCAN\_DataBits7Error*** Data bit 7 error in this byte.

***kFLEXCAN\_AllZeroError*** All-zeros non-correctable error in this byte.

***kFLEXCAN\_AllOneError*** All-ones non-correctable error in this byte.

***kFLEXCAN\_NonCorrectableErrors*** Non-correctable error in this byte.

## 16.2.6 Function Documentation

### 16.2.6.1 bool FLEXCAN\_IsInstanceHasFDMode ( CAN\_Type \* *base* )

Note

Use this API only if different soc parts share the SOC part name macro define. Otherwise, a different SOC part name can be used to determine at compile time whether the FlexCAN instance supports CAN FD mode or not. If need use this API to determine if CAN FD mode is supported, the FLEXCAN\_Init function needs to be executed first, and then call this API and use the return to value determines whether to supports CAN FD mode, if return true, continue calling FLEXCAN\_FDIInit to enable CAN FD mode.

Parameters

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

Returns

return TRUE if instance support CAN FD mode, FALSE if instance only support classic CAN (2.0) mode.

#### 16.2.6.2 void FLEXCAN\_EnterFreezeMode ( CAN\_Type \* *base* )

This function makes the FlexCAN work under Freeze Mode.

Parameters

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

#### 16.2.6.3 void FLEXCAN\_ExitFreezeMode ( CAN\_Type \* *base* )

This function makes the FlexCAN leave Freeze Mode.

Parameters

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

#### 16.2.6.4 uint32\_t FLEXCANGetInstance ( CAN\_Type \* *base* )

Parameters

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

Returns

FlexCAN instance.

#### 16.2.6.5 bool FLEXCAN\_CalculateImprovedTimingValues ( CAN\_Type \* *base*, uint32\_t *bitRate*, uint32\_t *sourceClock\_Hz*, flexcan\_timing\_config\_t \* *pTimingConfig* )

This function use to calculates the Classical CAN timing values according to the given bit rate. The Calculated timing values will be set in CTRL1/CBT/ENCBT register. The calculation is based on the recommendation of the CiA 301 v4.2.0 and previous version document.

## Parameters

|                       |                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                                                        |
| <i>bitRate</i>        | The classical CAN speed in bps defined by user, should be less than or equal to 1-Mbps. |
| <i>sourceClock_Hz</i> | The Source clock frequency in Hz.                                                       |
| <i>pTimingConfig</i>  | Pointer to the FlexCAN timing configuration structure.                                  |

## Returns

TRUE if timing configuration found, FALSE if failed to find configuration.

### 16.2.6.6 void FLEXCAN\_Init ( CAN\_Type \* *base*, const flexcan\_config\_t \* *pConfig*, uint32\_t *sourceClock\_Hz* )

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the `flexcan_config_t` parameters and how to call the `FLEXCAN_Init` function by passing in these parameters.

```
* flexcan_config_t flexcanConfig;
* flexcanConfig.clkSrc = kFLEXCAN_ClkSrc0;
* flexcanConfig.bitRate = 1000000U;
* flexcanConfig.maxMbNum = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeup = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.enableDoze = false;
* flexcanConfig.disableSelfReception = false;
* flexcanConfig.enableListenOnlyMode = false;
* flexcanConfig.timingConfig = timingConfig;
* FLEXCAN_Init(CAN0, &flexcanConfig, 40000000UL);
*
```

## Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                      |
| <i>pConfig</i>        | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |

### 16.2.6.7 **bool FLEXCAN\_FDCalculateImprovedTimingValues ( CAN\_Type \* *base*, uint32\_t *bitRate*, uint32\_t *bitRateFD*, uint32\_t *sourceClock\_Hz*, flexcan\_timing\_config\_t \* *pTimingConfig* )**

This function use to calculates the CANFD timing values according to the given nominal phase bit rate and data phase bit rate. The Calculated timing values will be set in CBT/ENCBT and FDCBT/EDCBT registers. The calculation is based on the recommendation of the CiA 1301 v1.0.0 document.

Parameters

|                       |                                                                                                        |
|-----------------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                                                                       |
| <i>bitRate</i>        | The CANFD bus control speed in bps defined by user.                                                    |
| <i>bitRateFD</i>      | The CAN FD data phase speed in bps defined by user. Equal to bitRate means disable bit rate switching. |
| <i>sourceClock_Hz</i> | The Source clock frequency in Hz.                                                                      |
| <i>pTimingConfig</i>  | Pointer to the FlexCAN timing configuration structure.                                                 |

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

### 16.2.6.8 **void FLEXCAN\_FDInit ( CAN\_Type \* *base*, const flexcan\_config\_t \* *pConfig*, uint32\_t *sourceClock\_Hz*, flexcan\_mb\_size\_t *dataSize*, bool *brs* )**

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the `flexcan_config_t` parameters and how to call the `FLEXCAN_FDInit` function by passing in these parameters.

```
* flexcan_config_t flexcanConfig;
* flexcanConfig.clkSrc = kFLEXCAN_ClkSrc0;
* flexcanConfig.bitRate = 1000000U;
* flexcanConfig.bitRateFD = 2000000U;
* flexcanConfig.maxMbNum = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeup = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.disableSelfReception = false;
* flexcanConfig.enableListenOnlyMode = false;
* flexcanConfig.enableDoze = false;
* flexcanConfig.timingConfig = timingConfig;
* FLEXCAN_FDInit(CANO0, &flexcanConfig, 80000000UL,
* kFLEXCAN_16BperMB, true);
*
```

Parameters

|                       |                                                                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                                                                                                             |
| <i>pConfig</i>        | Pointer to the user-defined configuration structure.                                                                                         |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz.                                                                                        |
| <i>dataSize</i>       | FlexCAN Message Buffer payload size. The actual transmitted or received CAN FD frame data size needs to be less than or equal to this value. |
| <i>brs</i>            | True if bit rate switch is enabled in FD mode.                                                                                               |

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

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

Parameters

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

#### 16.2.6.10 void FLEXCAN\_GetDefaultConfig ( flexcan\_config\_t \* *pConfig* )

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. *flexcanConfig->clkSrc* = kFLEXCAN\_ClkSrc0; *flexcanConfig->bitRate* = 1000000U; *flexcanConfig->bitRateFD* = 2000000U; *flexcanConfig->maxMbNum* = 16; *flexcanConfig->enableLoopBack* = false; *flexcanConfig->enableSelfWakeup* = false; *flexcanConfig->enableIndividMask* = false; *flexcanConfig->disableSelfReception* = false; *flexcanConfig->enableListenOnlyMode* = false; *flexcanConfig->enableDoze* = false; *flexcanConfig->enableMemoryErrorControl* = true; *flexcanConfig->enableNon-CorrectableErrorEnterFreeze* = true; *flexcanConfig.timingConfig* = timingConfig;

Parameters

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

#### 16.2.6.11 void FLEXCAN\_SetTimingConfig ( CAN\_Type \* *base*, const flexcan\_timing\_config\_t \* *pConfig* )

This function gives user settings to classical CAN or CAN FD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN\\_SetBitRate\(\)](#) instead.

## Note

Calling [FLEXCAN\\_SetTimingConfig\(\)](#) overrides the bit rate set in [FLEXCAN\\_Init\(\)](#) or [FLEXCAN\\_N\\_SetBitRate\(\)](#).

## Parameters

|                |                                                |
|----------------|------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.               |
| <i>pConfig</i> | Pointer to the timing configuration structure. |

### 16.2.6.12 status\_t FLEXCAN\_SetBitRate ( CAN\_Type \* *base*, uint32\_t *sourceClock\_Hz*, uint32\_t *bitRate\_Bps* )

This function set the bit rate of classical CAN frame or CAN FD frame nominal phase base on [FLEXCAN\\_CalculateImprovedTimingValues\(\)](#) API calculated timing values.

## Note

Calling [FLEXCAN\\_SetBitRate\(\)](#) overrides the bit rate set in [FLEXCAN\\_Init\(\)](#).

## Parameters

|                       |                                  |
|-----------------------|----------------------------------|
| <i>base</i>           | FlexCAN peripheral base address. |
| <i>sourceClock_Hz</i> | Source Clock in Hz.              |
| <i>bitRate_Bps</i>    | Bit rate in Bps.                 |

## Returns

kStatus\_Success - Set CAN baud rate (only Nominal phase) successfully.

### 16.2.6.13 void FLEXCAN\_SetFDTimingConfig ( CAN\_Type \* *base*, const flexcan\_timing\_config\_t \* *pConfig* )

This function gives user settings to CANFD data phase timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN\\_SetFDBitRate\(\)](#) to set both Nominal/-Data bit Rate instead.

## Note

Calling [FLEXCAN\\_SetFDTimingConfig\(\)](#) overrides the data phase bit rate set in [FLEXCAN\\_FD-Init\(\)](#)/[FLEXCAN\\_SetFDBitRate\(\)](#).

Parameters

|                |                                                |
|----------------|------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.               |
| <i>pConfig</i> | Pointer to the timing configuration structure. |

#### 16.2.6.14 status\_t FLEXCAN\_SetFDBitRate ( CAN\_Type \* *base*, uint32\_t *sourceClock\_Hz*, uint32\_t *bitRateN\_Bps*, uint32\_t *bitRateD\_Bps* )

This function set the baud rate of FLEXCAN FD base on [FLEXCAN\\_FDCalculateImprovedTimingValues\(\)](#) API calculated timing values.

Parameters

|                       |                                  |
|-----------------------|----------------------------------|
| <i>base</i>           | FlexCAN peripheral base address. |
| <i>sourceClock_Hz</i> | Source Clock in Hz.              |
| <i>bitRateN_Bps</i>   | Nominal bit Rate in Bps.         |
| <i>bitRateD_Bps</i>   | Data bit Rate in Bps.            |

Returns

kStatus\_Success - Set CAN FD bit rate (include Nominal and Data phase) successfully.

#### 16.2.6.15 void FLEXCAN\_SetRxMbGlobalMask ( CAN\_Type \* *base*, uint32\_t *mask* )

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the [FLEXCAN\\_Init\(\)](#).

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.     |
| <i>mask</i> | Rx Message Buffer Global Mask value. |

#### 16.2.6.16 void FLEXCAN\_SetRxFifoGlobalMask ( CAN\_Type \* *base*, uint32\_t *mask* )

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FlexCAN peripheral base address. |
| <i>mask</i> | Rx Fifo Global Mask value.       |

#### 16.2.6.17 void FLEXCAN\_SetRxIndividualMask ( CAN\_Type \* *base*, uint8\_t *maskIdx*, uint32\_t *mask* )

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the [FLEXCAN\\_Init\(\)](#). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>base</i>    | FlexCAN peripheral base address. |
| <i>maskIdx</i> | The Index of individual Mask.    |
| <i>mask</i>    | Rx Individual Mask value.        |

#### 16.2.6.18 void FLEXCAN\_SetTxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

|               |                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                          |
| <i>enable</i> | Enable/disable Tx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Tx Message Buffer.</li> <li>• false: Disable Tx Message Buffer.</li> </ul> |

#### 16.2.6.19 void FLEXCAN\_SetFDTxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

|               |                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                          |
| <i>enable</i> | Enable/disable Tx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Tx Message Buffer.</li> <li>• false: Disable Tx Message Buffer.</li> </ul> |

#### **16.2.6.20 void FLEXCAN\_SetRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_rx\_mb\_config\_t \* *pRxMbConfig*, bool *enable* )**

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

|                    |                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>       | The Message Buffer index.                                                                                                                                          |
| <i>pRxMbConfig</i> | Pointer to the FlexCAN Message Buffer configuration structure.                                                                                                     |
| <i>enable</i>      | Enable/disable Rx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Rx Message Buffer.</li> <li>• false: Disable Rx Message Buffer.</li> </ul> |

#### **16.2.6.21 void FLEXCAN\_SetFDRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_rx\_mb\_config\_t \* *pRxMbConfig*, bool *enable* )**

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

|                    |                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>       | The Message Buffer index.                                                                                                                                          |
| <i>pRxMbConfig</i> | Pointer to the FlexCAN Message Buffer configuration structure.                                                                                                     |
| <i>enable</i>      | Enable/disable Rx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Rx Message Buffer.</li> <li>• false: Disable Rx Message Buffer.</li> </ul> |

### **16.2.6.22 void FLEXCAN\_SetRxFifoConfig ( CAN\_Type \* *base*, const flexcan\_rx\_fifo\_config\_t \* *pRxFifoConfig*, bool *enable* )**

This function configures the FlexCAN Rx FIFO with given configuration.

Note

Legacy Rx FIFO only can receive classic CAN message.

Parameters

|                      |                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | FlexCAN peripheral base address.                                                                                                                          |
| <i>pRxFifoConfig</i> | Pointer to the FlexCAN Legacy Rx FIFO configuration structure. Can be NULL when enable parameter is false.                                                |
| <i>enable</i>        | Enable/disable Legacy Rx FIFO. <ul style="list-style-type: none"> <li>• true: Enable Legacy Rx FIFO.</li> <li>• false: Disable Legacy Rx FIFO.</li> </ul> |

### **16.2.6.23 void FLEXCAN\_SetEnhancedRxFifoConfig ( CAN\_Type \* *base*, const flexcan\_enhanced\_rx\_fifo\_config\_t \* *pConfig*, bool *enable* )**

This function configures the Enhanced Rx FIFO with given configuration.

Note

Enhanced Rx FIFO support receive classic CAN or CAN FD messages, Legacy Rx FIFO and Enhanced Rx FIFO cannot be enabled at the same time.

Parameters

|                |                                                                                                                                                                 |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                                                                                                                                |
| <i>pConfig</i> | Pointer to the FlexCAN Enhanced Rx FIFO configuration structure. Can be NULL when enable parameter is false.                                                    |
| <i>enable</i>  | Enable/disable Enhanced Rx FIFO. <ul style="list-style-type: none"> <li>• true: Enable Enhanced Rx FIFO.</li> <li>• false: Disable Enhanced Rx FIFO.</li> </ul> |

#### **16.2.6.24 static uint64\_t FLEXCAN\_GetStatusFlags ( CAN\_Type \* *base* ) [inline], [static]**

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators [\\_flexcan\\_flags](#). To check the specific status, compare the return value with enumerators in [\\_flexcan\\_flags](#).

Parameters

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

Returns

FlexCAN status flags which are ORed by the enumerators in the [\\_flexcan\\_flags](#).

#### **16.2.6.25 static void FLEXCAN\_ClearStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                           |
| <i>mask</i> | The status flags to be cleared, it is logical OR value of <a href="#">_flexcan_flags</a> . |

#### **16.2.6.26 static void FLEXCAN\_GetBusErrCount ( CAN\_Type \* *base*, uint8\_t \* *txErrBuf*, uint8\_t \* *rxErrBuf* ) [inline], [static]**

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>txErrBuf</i> | Buffer to store Tx Error Counter value. |
| <i>rxErrBuf</i> | Buffer to store Rx Error Counter value. |

#### **16.2.6.27 static uint64\_t FLEXCAN\_GetMbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

This function gets the interrupt flags of a given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

Returns

The status of given Message Buffers.

#### **16.2.6.28 static uint64\_t FLEXCAN\_GetHigh64MbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

Valid only if the number of available MBs exceeds 64.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

Returns

The status of given Message Buffers.

#### **16.2.6.29 static void FLEXCAN\_ClearMbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

This function clears the interrupt flags of a given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### **16.2.6.30 static void FLEXCAN\_ClearHigh64MbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

Valid only if the number of available MBs exceeds 64.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### **16.2.6.31 void FLEXCAN\_GetMemoryErrorReportStatus ( CAN\_Type \* *base*, flexcan\_memory\_error\_report\_status\_t \* *errorStatus* )**

This function gets the FlexCAN Memory Error Report registers status.

Parameters

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <i>base</i>        | FlexCAN peripheral base address.                                   |
| <i>errorStatus</i> | Pointer to FlexCAN Memory Error Report registers status structure. |

#### **16.2.6.32 static uint32\_t FLEXCAN\_GetEnhancedFifoDataCount ( CAN\_Type \* *base* ) [inline], [static]**

This function gets the number of CAN messages stored in the Enhanced Rx FIFO.

Parameters

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

Returns

The number of available CAN messages stored in the Enhanced Rx FIFO.

#### **16.2.6.33 static void FLEXCAN\_EnableInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

#### 16.2.6.34 static void FLEXCAN\_DisableInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

Parameters

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

#### 16.2.6.35 static void FLEXCAN\_EnableMbInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function enables the interrupts of given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### 16.2.6.36 static void FLEXCAN\_EnableHigh64MbInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

Valid only if the number of available MBs exceeds 64.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### 16.2.6.37 static void FLEXCAN\_DisableMbInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function disables the interrupts of given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### **16.2.6.38 static void FLEXCAN\_DisableHigh64MbInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

Valid only if the number of available MBs exceeds 64.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### **16.2.6.39 void FLEXCAN\_EnableRxFifoDMA ( CAN\_Type \* *base*, bool *enable* )**

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

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

#### **16.2.6.40 static uintptr\_t FLEXCAN\_GetRxFifoHeadAddr ( CAN\_Type \* *base* ) [inline], [static]**

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

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

Returns

FlexCAN Rx FIFO Head address.

#### **16.2.6.41 static void FLEXCAN\_Enable( CAN\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the FlexCAN module.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN base pointer.             |
| <i>enable</i> | true to enable, false to disable. |

#### 16.2.6.42 status\_t FLEXCAN\_WriteTxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_frame\_t \* *pTxFrame* )

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.         |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.        |
| <i>pTxFrame</i> | Pointer to CAN message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

#### 16.2.6.43 status\_t FLEXCAN\_ReadRxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pRxFrame* )

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                      |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.                     |
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

#### **16.2.6.44 status\_t FLEXCAN\_WriteFDTxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_fd\_frame\_t \* *pTxFrame* )**

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.            |
| <i>mbIdx</i>    | The FlexCAN FD Message Buffer index.        |
| <i>pTxFrame</i> | Pointer to CAN FD message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

#### **16.2.6.45 status\_t FLEXCAN\_ReadFDRxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *pRxFrame* )**

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                         |
| <i>mbIdx</i>    | The FlexCAN FD Message Buffer index.                     |
| <i>pRxFrame</i> | Pointer to CAN FD message frame structure for reception. |

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx_Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

#### 16.2.6.46 status\_t FLEXCAN\_ReadRxFifo ( CAN\_Type \* *base*, flexcan\_frame\_t \* *pRxFrame* )

This function reads a CAN message from the FlexCAN Legacy Rx FIFO.

Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                      |
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

#### 16.2.6.47 status\_t FLEXCAN\_ReadEnhancedRxFifo ( CAN\_Type \* *base*, flexcan\_fd\_frame\_t \* *pRxFrame* )

This function reads a CAN or CAN FD message from the FlexCAN Enhanced Rx FIFO.

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                         |
| <i>pRxFrame</i> | Pointer to CAN FD message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

#### 16.2.6.48 status\_t FLEXCAN\_TransferFDSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *pTxFrame* )

Note

A transfer handle does not need to be created before calling this API.

Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | FlexCAN peripheral base pointer.     |
| <i>mbIdx</i> | The FlexCAN FD Message Buffer index. |

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>pTxFrame</i> | Pointer to CAN FD message frame to be sent. |
|-----------------|---------------------------------------------|

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

#### 16.2.6.49 status\_t FLEXCAN\_TransferFDReceiveBlocking ( **CAN\_Type \* base**, **uint8\_t mbIdx**, **flexcan\_fd\_frame\_t \* pRxFrame** )

Note

A transfer handle does not need to be created before calling this API.

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.                         |
| <i>mbIdx</i>    | The FlexCAN FD Message Buffer index.                     |
| <i>pRxFrame</i> | Pointer to CAN FD message frame structure for reception. |

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

#### 16.2.6.50 status\_t FLEXCAN\_TransferFDSendNonBlocking ( **CAN\_Type \* base**, **flexcan\_handle\_t \* handle**, **flexcan\_mb\_transfer\_t \* pMbXfer** )

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

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

|                |                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------|
| <i>handle</i>  | FlexCAN handle pointer.                                                                       |
| <i>pMbXfer</i> | FlexCAN FD Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                       |
|--------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>         | Start Tx Message Buffer sending process successfully. |
| <i>kStatus_Fail</i>            | Write Tx Message Buffer failed.                       |
| <i>kStatus_FLEXCAN_Tx-Busy</i> | Tx Message Buffer is in use.                          |

#### 16.2.6.51 **status\_t FLEXCAN\_TransferFDReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )**

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

|                |                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                                                              |
| <i>handle</i>  | FlexCAN handle pointer.                                                                       |
| <i>pMbXfer</i> | FlexCAN FD Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                           |
|--------------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>         | - Start Rx Message Buffer receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-Busy</i> | - Rx Message Buffer is in use.                            |

#### 16.2.6.52 **void FLEXCAN\_TransferFDAbortSend ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )**

This function aborts the interrupt driven message send process.

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.     |
| <i>handle</i> | FlexCAN handle pointer.              |
| <i>mbIdx</i>  | The FlexCAN FD Message Buffer index. |

**16.2.6.53 void FLEXCAN\_TransferFDAbortReceive ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )**

This function aborts the interrupt driven message receive process.

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.     |
| <i>handle</i> | FlexCAN handle pointer.              |
| <i>mbIdx</i>  | The FlexCAN FD Message Buffer index. |

#### 16.2.6.54 status\_t FLEXCAN\_TransferSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pTxFrame* )

Note

A transfer handle does not need to be created before calling this API.

Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.         |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.        |
| <i>pTxFrame</i> | Pointer to CAN message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

#### 16.2.6.55 status\_t FLEXCAN\_TransferReceiveBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pRxFrame* )

Note

A transfer handle does not need to be created before calling this API.

Parameters

|              |                                   |
|--------------|-----------------------------------|
| <i>base</i>  | FlexCAN peripheral base pointer.  |
| <i>mbIdx</i> | The FlexCAN Message Buffer index. |

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |
|-----------------|-------------------------------------------------------|

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx_Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

#### 16.2.6.56 status\_t FLEXCAN\_TransferReceiveFifoBlocking ( CAN\_Type \* *base*, flexcan\_frame\_t \* *pRxFrame* )

Note

A transfer handle does not need to be created before calling this API.

Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.                      |
| <i>pRxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

#### 16.2.6.57 status\_t FLEXCAN\_TransferReceiveEnhancedFifoBlocking ( CAN\_Type \* *base*, flexcan\_fd\_frame\_t \* *pRxFrame* )

Note

A transfer handle does not need to be created before calling this API.

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.                         |
| <i>pRxFrame</i> | Pointer to CAN FD message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

#### 16.2.6.58 void FLEXCAN\_TransferCreateHandle ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_transfer\_callback\_t *callback*, void \* *userData* )

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

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>handle</i>   | FlexCAN handle pointer.                 |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

#### 16.2.6.59 status\_t FLEXCAN\_TransferSendNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                                                           |
| <i>handle</i>  | FlexCAN handle pointer.                                                                    |
| <i>pMbXfer</i> | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                       |
|--------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>         | Start Tx Message Buffer sending process successfully. |
| <i>kStatus_Fail</i>            | Write Tx Message Buffer failed.                       |
| <i>kStatus_FLEXCAN_Tx-Busy</i> | Tx Message Buffer is in use.                          |

**16.2.6.60 status\_t FLEXCAN\_TransferReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )**

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                                                           |
| <i>handle</i>  | FlexCAN handle pointer.                                                                    |
| <i>pMbXfer</i> | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                           |
|--------------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>         | - Start Rx Message Buffer receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-Busy</i> | - Rx Message Buffer is in use.                            |

#### 16.2.6.61 status\_t FLEXCAN\_TransferReceiveFifoNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_fifo\_transfer\_t \* *pFifoXfer* )

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

|                  |                                                                                       |
|------------------|---------------------------------------------------------------------------------------|
| <i>base</i>      | FlexCAN peripheral base address.                                                      |
| <i>handle</i>    | FlexCAN handle pointer.                                                               |
| <i>pFifoXfer</i> | FlexCAN Rx FIFO transfer structure. See the <a href="#">flexcan_fifo_transfer_t</a> . |

Return values

|                                    |                                                 |
|------------------------------------|-------------------------------------------------|
| <i>kStatus_Success</i>             | - Start Rx FIFO receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | - Rx FIFO is currently in use.                  |

#### 16.2.6.62 status\_t FLEXCAN\_TransferGetReceiveFifoCount ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                       |
| <i>handle</i> | FlexCAN handle pointer.                                                |
| <i>count</i>  | Number of CAN messages receive so far by the non-blocking transaction. |

Return values

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

#### 16.2.6.63 **status\_t FLEXCAN\_TransferReceiveEnhancedFifoNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_fifo\_transfer\_t \* *pFifoXfer* )**

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

|                  |                                                                                            |
|------------------|--------------------------------------------------------------------------------------------|
| <i>base</i>      | FlexCAN peripheral base address.                                                           |
| <i>handle</i>    | FlexCAN handle pointer.                                                                    |
| <i>pFifoXfer</i> | FlexCAN Rx FIFO transfer structure. See the ref <a href="#">flexcan_fifo_transfer_t</a> .@ |

Return values

|                                    |                                                 |
|------------------------------------|-------------------------------------------------|
| <i>kStatus_Success</i>             | - Start Rx FIFO receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | - Rx FIFO is currently in use.                  |

#### 16.2.6.64 **static status\_t FLEXCAN\_TransferGetReceiveEnhancedFifoCount ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                       |
| <i>handle</i> | FlexCAN handle pointer.                                                |
| <i>count</i>  | Number of CAN messages receive so far by the non-blocking transaction. |

Return values

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

#### 16.2.6.65 `uint32_t FLEXCAN_GetTimeStamp ( flexcan_handle_t * handle, uint8_t mbIdx )`

Then function can only be used when calling non-blocking Data transfer (TX/RX) API, After TX/RX data transfer done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle, Detail non-blocking data transfer API (TX/RX) contain. -FLEXCAN\_TransferSendNonBlocking -FLEXCAN\_TransferFDSendNonBlocking -FLEXCAN\_TransferReceiveNonBlocking -FLEXCAN\_TransferFDReceiveNonBlocking -FLEXCAN\_TransferReceiveFifoNonBlocking

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mbIdx</i>  | The FlexCAN Message Buffer index. |

Return values

|            |                                                     |
|------------|-----------------------------------------------------|
| <i>the</i> | index of mailbox 's timestamp stored in the handle. |
|------------|-----------------------------------------------------|

#### 16.2.6.66 `void FLEXCAN_TransferAbortSend ( CAN_Type * base, flexcan_handle_t * handle, uint8_t mbIdx )`

This function aborts the interrupt driven message send process.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mbIdx</i>  | The FlexCAN Message Buffer index. |

#### 16.2.6.67 `void FLEXCAN_TransferAbortReceive ( CAN_Type * base, flexcan_handle_t * handle, uint8_t mbIdx )`

This function aborts the interrupt driven message receive process.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mbIdx</i>  | The FlexCAN Message Buffer index. |

#### **16.2.6.68 void FLEXCAN\_TransferAbortReceiveFifo ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )**

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

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

#### **16.2.6.69 void FLEXCAN\_TransferAbortReceiveEnhancedFifo ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )**

This function aborts the interrupt driven message receive from Enhanced Rx FIFO process.

Parameters

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

#### **16.2.6.70 void FLEXCAN\_TransferHandleIRQ ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )**

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

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

# Chapter 17

## FLEXSPI: Flexible Serial Peripheral Interface Driver

### 17.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flexible Serial Peripheral Interface (FLEXSPI) module of MCUXpresso SDK/i.MX devices.

FLEXSPI driver includes functional APIs and interrupt/EDMA non-blocking transactional APIs.

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

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `flexspi_handle_t`/`flexspi_edma_handle_t` as the second parameter. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandle` API. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandleEDMA` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXSPI_TransferNonBlocking()` and `FLEXSPI_TransferEDMA()` set up data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXSPI_Idle` status.

### Data Structures

- struct `flexspi_config_t`  
*FLEXSPI configuration structure. [More...](#)*
- struct `flexspi_device_config_t`  
*External device configuration items. [More...](#)*
- struct `flexspi_transfer_t`  
*Transfer structure for FLEXSPI. [More...](#)*
- struct `flexspi_handle_t`  
*Transfer handle structure for FLEXSPI. [More...](#)*

### Macros

- #define `FLEXSPI_LUT_SEQ`(cmd0, pad0, op0, cmd1, pad1, op1)  
*Formula to form FLEXSPI instructions in LUT table.*

## Typedefs

- `typedef void(* flexspi_transfer_callback_t )(FLEXSPI_Type *base, flexspi_handle_t *handle, status_t status, void *userData)`  
*FLEXSPI transfer callback function.*

## Enumerations

- `enum {`  
 `kStatus_FLEXSPI_Busy = MAKE_STATUS(kStatusGroup_FLEXSPI, 0),`  
 `kStatus_FLEXSPI_SequenceExecutionTimeout = MAKE_STATUS(kStatusGroup_FLEXSPI, 1),`  
 `kStatus_FLEXSPI_IpCommandSequenceError = MAKE_STATUS(kStatusGroup_FLEXSPI, 2),`  
 `kStatus_FLEXSPI_IpCommandGrantTimeout = MAKE_STATUS(kStatusGroup_FLEXSPI, 3) }`  
*Status structure of FLEXSPI.*
- `enum {`  
 `kFLEXSPI_Command_STOP = 0x00U,`  
 `kFLEXSPI_Command_SDR = 0x01U,`  
 `kFLEXSPI_Command_RADDR_SDR = 0x02U,`  
 `kFLEXSPI_Command_CADDR_SDR = 0x03U,`  
 `kFLEXSPI_Command_MODE1_SDR = 0x04U,`  
 `kFLEXSPI_Command_MODE2_SDR = 0x05U,`  
 `kFLEXSPI_Command_MODE4_SDR = 0x06U,`  
 `kFLEXSPI_Command_MODE8_SDR = 0x07U,`  
 `kFLEXSPI_Command_WRITE_SDR = 0x08U,`  
 `kFLEXSPI_Command_READ_SDR = 0x09U,`  
 `kFLEXSPI_Command_LEARN_SDR = 0x0AU,`  
 `kFLEXSPI_Command_DATSZ_SDR = 0x0BU,`  
 `kFLEXSPI_Command_DUMMY_SDR = 0x0CU,`  
 `kFLEXSPI_Command_DUMMY_RWDS_SDR = 0x0DU,`  
 `kFLEXSPI_Command_DDR = 0x21U,`  
 `kFLEXSPI_Command_RADDR_DDR = 0x22U,`  
 `kFLEXSPI_Command_CADDR_DDR = 0x23U,`  
 `kFLEXSPI_Command_MODE1_DDR = 0x24U,`  
 `kFLEXSPI_Command_MODE2_DDR = 0x25U,`  
 `kFLEXSPI_Command_MODE4_DDR = 0x26U,`  
 `kFLEXSPI_Command_MODE8_DDR = 0x27U,`  
 `kFLEXSPI_Command_WRITE_DDR = 0x28U,`  
 `kFLEXSPI_Command_READ_DDR = 0x29U,`  
 `kFLEXSPI_Command_LEARN_DDR = 0x2AU,`  
 `kFLEXSPI_Command_DATSZ_DDR = 0x2BU,`  
 `kFLEXSPI_Command_DUMMY_DDR = 0x2CU,`  
 `kFLEXSPI_Command_DUMMY_RWDS_DDR = 0x2DU,`  
 `kFLEXSPI_Command_JUMP_ON_CS = 0x1FU }`  
*CMD definition of FLEXSPI, use to form LUT instruction, \_flexspi\_command.*
- `enum flexspi_pad_t {`

```
kFLEXSPI_1PAD = 0x00U,
kFLEXSPI_2PAD = 0x01U,
kFLEXSPI_4PAD = 0x02U,
kFLEXSPI_8PAD = 0x03U }
```

*pad definition of FLEXSPI, use to form LUT instruction.*

- enum `flexspi_flags_t` {
   
kFLEXSPI\_SequenceExecutionTimeoutFlag = FLEXSPI\_INTEN\_SEQTIMEOUTEN\_MASK,
 kFLEXSPI\_AhbBusErrorFlag = FLEXSPI\_INTEN\_AHBBUSERROREN\_MASK,
 kFLEXSPI\_SckStoppedBecauseTxEmptyFlag,
 kFLEXSPI\_SckStoppedBecauseRxFullFlag,
 kFLEXSPI\_IpTxFifoWatermarkEmptyFlag = FLEXSPI\_INTEN\_IPTXWEEN\_MASK,
 kFLEXSPI\_IpRxFifoWatermarkAvailableFlag = FLEXSPI\_INTEN\_IPRXWAEN\_MASK,
 kFLEXSPI\_AhbCommandSequenceErrorFlag,
 kFLEXSPI\_IpCommandSequenceErrorFlag = FLEXSPI\_INTEN\_IPCMDERREN\_MASK,
 kFLEXSPI\_AhbCommandGrantTimeoutFlag,
 kFLEXSPI\_IpCommandGrantTimeoutFlag,
 kFLEXSPI\_IpCommandExecutionDoneFlag,
 kFLEXSPI\_AllInterruptFlags = 0xFFFFU }

*FLEXSPI interrupt status flags.*

- enum `flexspi_read_sample_clock_t` {
   
kFLEXSPI\_ReadSampleClkLoopbackInternally = 0x0U,
 kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad = 0x1U,
 kFLEXSPI\_ReadSampleClkLoopbackFromSckPad = 0x2U,
 kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad = 0x3U }

*FLEXSPI sample clock source selection for Flash Reading.*

- enum `flexspi_cs_interval_cycle_unit_t` {
   
kFLEXSPI\_CsIntervalUnit1SckCycle = 0x0U,
 kFLEXSPI\_CsIntervalUnit256SckCycle = 0x1U }

*FLEXSPI interval unit for flash device select.*

- enum `flexspi_ahb_write_wait_unit_t` {
   
kFLEXSPI\_AhbWriteWaitUnit2AhbCycle = 0x0U,
 kFLEXSPI\_AhbWriteWaitUnit8AhbCycle = 0x1U,
 kFLEXSPI\_AhbWriteWaitUnit32AhbCycle = 0x2U,
 kFLEXSPI\_AhbWriteWaitUnit128AhbCycle = 0x3U,
 kFLEXSPI\_AhbWriteWaitUnit512AhbCycle = 0x4U,
 kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle = 0x5U,
 kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle = 0x6U,
 kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle = 0x7U }

*FLEXSPI AHB wait interval unit for writing.*

- enum `flexspi_ip_error_code_t` {

```

kFLEXSPI_IpCmdErrorNoError = 0x0U,
kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd = 0x2U,
kFLEXSPI_IpCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_IpCmdErrorInvalidAddress = 0x6U,
kFLEXSPI_IpCmdErrorSequenceExecutionTimeout = 0xEU,
kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss = 0xFU }

```

*Error Code when IP command Error detected.*

- enum `flexspi_ahb_error_code_t` {
 

```

kFLEXSPI_AhbCmdErrorNoError = 0x0U,
kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd = 0x2U,
kFLEXSPI_AhbCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_AhbCmdSequenceExecutionTimeout = 0x6U }
```

*Error Code when AHB command Error detected.*

- enum `flexspi_port_t` {
 

```

kFLEXSPI_PortA1 = 0x0U,
kFLEXSPI_PortA2,
kFLEXSPI_PortB1,
kFLEXSPI_PortB2 }
```
- enum `flexspi_arb_command_source_t`

*FLEXSPI operation port select.*
- enum `flexspi_command_type_t`

*Trigger source of current command sequence granted by arbitrator.*

```

kFLEXSPI_Command,
kFLEXSPI_Config }
```

*Command type.*

## Driver version

- #define `FSL_FLEXSPI_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)  
*FLEXSPI driver version.*

## Initialization and deinitialization

- `uint32_t FLEXSPIGetInstance (FLEXSPI_Type *base)`  
*Get the instance number for FLEXSPI.*
- `status_t FLEXSPI_CheckAndClearError (FLEXSPI_Type *base, uint32_t status)`  
*Check and clear IP command execution errors.*
- `void FLEXSPI_Init (FLEXSPI_Type *base, const flexspi_config_t *config)`  
*Initializes the FLEXSPI module and internal state.*
- `void FLEXSPI_GetDefaultConfig (flexspi_config_t *config)`  
*Gets default settings for FLEXSPI.*
- `void FLEXSPI_Deinit (FLEXSPI_Type *base)`  
*Deinitializes the FLEXSPI module.*

- void **FLEXSPI\_UpdateDllValue** (FLEXSPI\_Type \*base, **flexspi\_device\_config\_t** \*config, **flexspi\_port\_t** port)  
*Update FLEXSPI DLL value depending on currently flexspi root clock.*
- void **FLEXSPI\_SetFlashConfig** (FLEXSPI\_Type \*base, **flexspi\_device\_config\_t** \*config, **flexspi\_port\_t** port)  
*Configures the connected device parameter.*
- static void **FLEXSPI\_SoftwareReset** (FLEXSPI\_Type \*base)  
*Software reset for the FLEXSPI logic.*
- static void **FLEXSPI\_Enable** (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables the FLEXSPI module.*

## Interrupts

- static void **FLEXSPI\_EnableInterrupts** (FLEXSPI\_Type \*base, uint32\_t mask)  
*Enables the FLEXSPI interrupts.*
- static void **FLEXSPI\_DisableInterrupts** (FLEXSPI\_Type \*base, uint32\_t mask)  
*Disable the FLEXSPI interrupts.*

## DMA control

- static void **FLEXSPI\_EnableTxDMA** (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Tx FIFO DMA requests.*
- static void **FLEXSPI\_EnableRxDMA** (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Rx FIFO DMA requests.*
- static uint32\_t **FLEXSPI\_GetTxFifoAddress** (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP tx fifo address for DMA transfer.*
- static uint32\_t **FLEXSPI\_GetRxFifoAddress** (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP rx fifo address for DMA transfer.*

## FIFO control

- static void **FLEXSPI\_ResetFifos** (FLEXSPI\_Type \*base, bool txFifo, bool rxFifo)  
*Clears the FLEXSPI IP FIFO logic.*
- static void **FLEXSPI\_GetFifoCounts** (FLEXSPI\_Type \*base, size\_t \*txCount, size\_t \*rxCount)  
*Gets the valid data entries in the FLEXSPI FIFOs.*

## Status

- static uint32\_t **FLEXSPI\_GetInterruptStatusFlags** (FLEXSPI\_Type \*base)  
*Get the FLEXSPI interrupt status flags.*
- static void **FLEXSPI\_ClearInterruptStatusFlags** (FLEXSPI\_Type \*base, uint32\_t mask)  
*Get the FLEXSPI interrupt status flags.*
- static **flexspi\_arb\_command\_source\_t** **FLEXSPI\_GetArbitratorCommandSource** (FLEXSPI\_Type \*base)  
*Gets the trigger source of current command sequence granted by arbitrator.*
- static **flexspi\_ip\_error\_code\_t** **FLEXSPI\_GetIPCommandErrorCode** (FLEXSPI\_Type \*base, uint8\_t \*index)  
*Gets the error code when IP command error detected.*
- static **flexspi\_ahb\_error\_code\_t** **FLEXSPI\_GetAHBCommandErrorCode** (FLEXSPI\_Type \*base, uint8\_t \*index)

- Gets the error code when AHB command error detected.
- static bool [FLEXSPI\\_GetBusIdleStatus](#) (FLEXSPI\_Type \*base)  
Returns whether the bus is idle.

## Bus Operations

- void [FLEXSPI\\_UpdateRxSampleClock](#) (FLEXSPI\_Type \*base, [flexspi\\_read\\_sample\\_clock\\_t](#) clockSource)  
*Update read sample clock source.*
- static void [FLEXSPI\\_EnableIPParallelMode](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables/disables the FLEXSPI IP command parallel mode.*
- static void [FLEXSPI\\_EnableAHBParallelMode](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables/disables the FLEXSPI AHB command parallel mode.*
- void [FLEXSPI\\_UpdateLUT](#) (FLEXSPI\_Type \*base, uint32\_t index, const uint32\_t \*cmd, uint32\_t count)  
*Updates the LUT table.*
- static void [FLEXSPI\\_WriteData](#) (FLEXSPI\_Type \*base, uint32\_t data, uint8\_t fifoIndex)  
*Writes data into FIFO.*
- static uint32\_t [FLEXSPI\\_ReadData](#) (FLEXSPI\_Type \*base, uint8\_t fifoIndex)  
*Receives data from data FIFO.*
- status\_t [FLEXSPI\\_WriteBlocking](#) (FLEXSPI\_Type \*base, uint8\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes using blocking method.*
- status\_t [FLEXSPI\\_ReadBlocking](#) (FLEXSPI\_Type \*base, uint8\_t \*buffer, size\_t size)  
*Receives a buffer of data bytes using a blocking method.*
- status\_t [FLEXSPI\\_TransferBlocking](#) (FLEXSPI\_Type \*base, [flexspi\\_transfer\\_t](#) \*xfer)  
*Execute command to transfer a buffer data bytes using a blocking method.*

## Transactional

- void [FLEXSPI\\_TransferCreateHandle](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle, [flexspi\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the FLEXSPI handle which is used in transactional functions.*
- status\_t [FLEXSPI\\_TransferNonBlocking](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle, [flexspi\\_transfer\\_t](#) \*xfer)  
*Performs a interrupt non-blocking transfer on the FLEXSPI bus.*
- status\_t [FLEXSPI\\_TransferGetCount](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- void [FLEXSPI\\_TransferAbort](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void [FLEXSPI\\_TransferHandleIRQ](#) (FLEXSPI\_Type \*base, [flexspi\\_handle\\_t](#) \*handle)  
*Master interrupt handler.*

## 17.2 Data Structure Documentation

### 17.2.1 struct [flexspi\\_config\\_t](#)

#### Data Fields

- [flexspi\\_read\\_sample\\_clock\\_t](#) rxSampleClock

- Sample Clock source selection for Flash Reading.
- bool `enableSckFreeRunning`  
Enable/disable SCK output free-running.
- bool `enableCombination`  
Enable/disable combining PORT A and B Data Pins (*SIOA[3:0] and SIOB[3:0]*) to support Flash Octal mode.
- bool `enableDoze`  
Enable/disable doze mode support.
- bool `enableHalfSpeedAccess`  
Enable/disable divide by 2 of the clock for half speed commands.
- bool `enableSckBDiffOpt`  
Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.
- bool `enableSameConfigForAll`  
Enable/disable same configuration for all connected devices when enabled, same configuration in *FLASHA1CRx* is applied to all.
- uint16\_t `seqTimeoutCycle`  
Timeout wait cycle for command sequence execution, timeout after *ahbGrantTimeoutCycle\*1024 serial root clock cycles*.
- uint8\_t `ipGrantTimeoutCycle`  
Timeout wait cycle for IP command grant, timeout after *ipGrantTimeoutCycle\*1024 AHB clock cycles*.
- uint8\_t `txWatermark`  
FLEXSPI IP transmit watermark value.
- uint8\_t `rxWatermark`  
FLEXSPI receive watermark value.
- bool `enableAHBWriteIpTxFifo`  
Enable AHB bus write access to IP TX FIFO.
- bool `enableAHBWriteIpRxFifo`  
Enable AHB bus write access to IP RX FIFO.
- uint8\_t `ahbGrantTimeoutCycle`  
Timeout wait cycle for AHB command grant, timeout after *ahbGrantTimeoutCycle\*1024 AHB clock cycles*.
- uint16\_t `ahbBusTimeoutCycle`  
Timeout wait cycle for AHB read/write access, timeout after *ahbBusTimeoutCycle\*1024 AHB clock cycles*.
- uint8\_t `resumeWaitCycle`  
Wait cycle for idle state before suspended command sequence resume, timeout after *ahbBusTimeoutCycle AHB clock cycles*.
- flexspi\_ahbBuffer\_config\_t `buffer` [FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNT]  
AHB buffer size.
- bool `enableClearAHBBufferOpt`  
Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.
- bool `enableReadAddressOpt`  
Enable/disable remove AHB read burst start address alignment limitation.
- bool `enableAHBPrefetch`  
Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.
- bool `enableAHBBufferable`  
Enable/disable AHB bufferable write access support, when enabled,

- FLEXSPI return before waiting for command execution finished.*
- **bool enableAHBCachable**  
*Enable AHB bus cachable read access support.*

## Field Documentation

- (1) **flexspi\_read\_sample\_clock\_t flexspi\_config\_t::rxSampleClock**
- (2) **bool flexspi\_config\_t::enableSckFreeRunning**
- (3) **bool flexspi\_config\_t::enableCombination**
- (4) **bool flexspi\_config\_t::enableDoze**
- (5) **bool flexspi\_config\_t::enableHalfSpeedAccess**
- (6) **bool flexspi\_config\_t::enableSckBDiffOpt**
- (7) **bool flexspi\_config\_t::enableSameConfigForAll**
- (8) **uint16\_t flexspi\_config\_t::seqTimeoutCycle**
- (9) **uint8\_t flexspi\_config\_t::ipGrantTimeoutCycle**
- (10) **uint8\_t flexspi\_config\_t::txWatermark**
- (11) **uint8\_t flexspi\_config\_t::rxWatermark**
- (12) **bool flexspi\_config\_t::enableAHBWriteIpTxFifo**
- (13) **bool flexspi\_config\_t::enableAHBWriteIpRxFifo**
- (14) **uint8\_t flexspi\_config\_t::ahbGrantTimeoutCycle**
- (15) **uint16\_t flexspi\_config\_t::ahbBusTimeoutCycle**
- (16) **uint8\_t flexspi\_config\_t::resumeWaitCycle**
- (17) **flexspi\_ahbBuffer\_config\_t flexspi\_config\_t::buffer[FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNT]**
- (18) **bool flexspi\_config\_t::enableClearAHBBufferOpt**
- (19) **bool flexspi\_config\_t::enableReadAddressOpt**

when enable, there is no AHB read burst start address alignment limitation.

- (20) **bool flexspi\_config\_t::enableAHBPrefetch**
- (21) **bool flexspi\_config\_t::enableAHBBufferable**
- (22) **bool flexspi\_config\_t::enableAHBCachable**

### 17.2.2 struct flexspi\_device\_config\_t

#### Data Fields

- **uint32\_t flexspiRootClk**  
*FLEXSPI serial root clock.*
- **bool isSck2Enabled**  
*FLEXSPI use SCK2.*
- **uint32\_t flashSize**  
*Flash size in KByte.*
- **flexspi\_cs\_interval\_cycle\_unit\_t CSIntervalUnit**  
*CS interval unit, 1 or 256 cycle.*
- **uint16\_t CSInterval**  
*CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.*
- **uint8\_t CSHoldTime**  
*CS line hold time.*
- **uint8\_t CSSetupTime**  
*CS line setup time.*
- **uint8\_t dataValidTime**  
*Data valid time for external device.*
- **uint8\_t columnspace**  
*Column space size.*
- **bool enableWordAddress**  
*If enable word address.*
- **uint8\_t AWRSeqIndex**  
*Sequence ID for AHB write command.*
- **uint8\_t AWRSeqNumber**  
*Sequence number for AHB write command.*
- **uint8\_t ARDSeqIndex**  
*Sequence ID for AHB read command.*
- **uint8\_t ARDSeqNumber**  
*Sequence number for AHB read command.*
- **flexspi\_ahb\_write\_wait\_unit\_t AHBWriteWaitUnit**  
*AHB write wait unit.*
- **uint16\_t AHBWriteWaitInterval**  
*AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.*
- **bool enableWriteMask**  
*Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.*

**Field Documentation**

- (1) `uint32_t flexspi_device_config_t::flexspiRootClk`
- (2) `bool flexspi_device_config_t::isSck2Enabled`
- (3) `uint32_t flexspi_device_config_t::flashSize`
- (4) `flexspi_cs_interval_cycle_unit_t flexspi_device_config_t::CSIntervalUnit`
- (5) `uint16_t flexspi_device_config_t::CSInterval`
- (6) `uint8_t flexspi_device_config_t::CSHoldTime`
- (7) `uint8_t flexspi_device_config_t::CSSetupTime`
- (8) `uint8_t flexspi_device_config_t::dataValidTime`
- (9) `uint8_t flexspi_device_config_t::columnspace`
- (10) `bool flexspi_device_config_t::enableWordAddress`
- (11) `uint8_t flexspi_device_config_t::AWRSeqIndex`
- (12) `uint8_t flexspi_device_config_t::AWRSeqNumber`
- (13) `uint8_t flexspi_device_config_t::ARDSeqIndex`
- (14) `uint8_t flexspi_device_config_t::ARDSeqNumber`
- (15) `flexspi_ahb_write_wait_unit_t flexspi_device_config_t::AHBWriteWaitUnit`
- (16) `uint16_t flexspi_device_config_t::AHBWriteWaitInterval`
- (17) `bool flexspi_device_config_t::enableWriteMask`

**17.2.3 struct flexspi\_transfer\_t****Data Fields**

- `uint32_t deviceAddress`  
*Operation device address.*
- `flexspi_port_t port`  
*Operation port.*
- `flexspi_command_type_t cmdType`  
*Execution command type.*
- `uint8_t seqIndex`  
*Sequence ID for command.*
- `uint8_t SeqNumber`  
*Sequence number for command.*

- `uint32_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Data size in bytes.*

### Field Documentation

- (1) `uint32_t flexspi_transfer_t::deviceAddress`
- (2) `flexspi_port_t flexspi_transfer_t::port`
- (3) `flexspi_command_type_t flexspi_transfer_t::cmdType`
- (4) `uint8_t flexspi_transfer_t::seqIndex`
- (5) `uint8_t flexspi_transfer_t::SeqNumber`
- (6) `uint32_t* flexspi_transfer_t::data`
- (7) `size_t flexspi_transfer_t::dataSize`

### 17.2.4 struct \_flexspi\_handle

#### Data Fields

- `uint32_t state`  
*Internal state for FLEXSPI transfer.*
- `uint8_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Remaining Data size in bytes.*
- `size_t transferTotalSize`  
*Total Data size in bytes.*
- `flexspi_transfer_callback_t completionCallback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*FLEXSPI callback function parameter.*

### Field Documentation

- (1) `uint8_t* flexspi_handle_t::data`
- (2) `size_t flexspi_handle_t::dataSize`
- (3) `size_t flexspi_handle_t::transferTotalSize`
- (4) `void* flexspi_handle_t::userData`

## 17.3 Macro Definition Documentation

**17.3.1 #define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))**

**17.3.2 #define FLEXSPI\_LUT\_SEQ( cmd0, pad0, op0, cmd1, pad1, op1 )**

**Value:**

```
(FLEXSPI_LUT_OPERAND0 (op0) | FLEXSPI_LUT_NUM_PADS0 (pad0) | FLEXSPI_LUT_OPCODE0 (cmd0) | FLEXSPI_LUT_OPERAND1
 (op1) | \
 FLEXSPI_LUT_NUM_PADS1 (pad1) | FLEXSPI_LUT_OPCODE1 (cmd1))
```

## 17.4 Typedef Documentation

**17.4.1 typedef void(\* flexspi\_transfer\_callback\_t)(FLEXSPI\_Type \*base,  
flexspi\_handle\_t \*handle, status\_t status, void \*userData)**

## 17.5 Enumeration Type Documentation

### 17.5.1 anonymous enum

Enumerator

**kStatus\_FLEXSPI\_Busy** FLEXSPI is busy.

**kStatus\_FLEXSPI\_SequenceExecutionTimeout** Sequence execution timeout error occurred during FLEXSPI transfer.

**kStatus\_FLEXSPI\_IpCommandSequenceError** IP command Sequence execution timeout error occurred during FLEXSPI transfer.

**kStatus\_FLEXSPI\_IpCommandGrantTimeout** IP command grant timeout error occurred during FLEXSPI transfer.

### 17.5.2 anonymous enum

Enumerator

**kFLEXSPI\_Command\_STOP** Stop execution, deassert CS.

**kFLEXSPI\_Command\_SDR** Transmit Command code to Flash, using SDR mode.

**kFLEXSPI\_Command\_RADDR\_SDR** Transmit Row Address to Flash, using SDR mode.

**kFLEXSPI\_Command\_CADDR\_SDR** Transmit Column Address to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE1\_SDR** Transmit 1-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE2\_SDR** Transmit 2-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE4\_SDR** Transmit 4-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE8\_SDR** Transmit 8-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_WRITE\_SDR** Transmit Programming Data to Flash, using SDR mode.

**kFLEXSPI\_Command\_READ\_SDR** Receive Read Data from Flash, using SDR mode.

***kFLEXSPI\_Command\_LEARN\_SDR*** Receive Read Data or Preamble bit from Flash, SDR mode.

***kFLEXSPI\_Command\_DATSZ\_SDR*** Transmit Read/Program Data size (byte) to Flash, SDR mode.

***kFLEXSPI\_Command\_DUMMY\_SDR*** Leave data lines undriven by FlexSPI controller.

***kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

***kFLEXSPI\_Command\_DDR*** Transmit Command code to Flash, using DDR mode.

***kFLEXSPI\_Command\_RADDR\_DDR*** Transmit Row Address to Flash, using DDR mode.

***kFLEXSPI\_Command\_CADDR\_DDR*** Transmit Column Address to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE1\_DDR*** Transmit 1-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE2\_DDR*** Transmit 2-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE4\_DDR*** Transmit 4-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE8\_DDR*** Transmit 8-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_WRITE\_DDR*** Transmit Programming Data to Flash, using DDR mode.

***kFLEXSPI\_Command\_READ\_DDR*** Receive Read Data from Flash, using DDR mode.

***kFLEXSPI\_Command\_LEARN\_DDR*** Receive Read Data or Preamble bit from Flash, DDR mode.

***kFLEXSPI\_Command\_DATSZ\_DDR*** Transmit Read/Program Data size (byte) to Flash, DDR mode.

***kFLEXSPI\_Command\_DUMMY\_DDR*** Leave data lines undriven by FlexSPI controller.

***kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

***kFLEXSPI\_Command\_JUMP\_ON\_CS*** Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence.

### 17.5.3 enum flexspi\_pad\_t

Enumerator

***kFLEXSPI\_1PAD*** Transmit command/address and transmit/receive data only through DATA0/DATA1.

***kFLEXSPI\_2PAD*** Transmit command/address and transmit/receive data only through DATA[1:0].

***kFLEXSPI\_4PAD*** Transmit command/address and transmit/receive data only through DATA[3:0].

***kFLEXSPI\_8PAD*** Transmit command/address and transmit/receive data only through DATA[7:0].

### 17.5.4 enum flexspi\_flags\_t

Enumerator

***kFLEXSPI\_SequenceExecutionTimeoutFlag*** Sequence execution timeout.

***kFLEXSPI\_AhbBusErrorFlag*** AHB Bus error flag.

***kFLEXSPI\_SckStoppedBecauseTxEmptyFlag*** SCK is stopped during command sequence because Async TX FIFO empty.

***kFLEXSPI\_SckStoppedBecauseRxFullFlag*** SCK is stopped during command sequence because Async RX FIFO full.

***kFLEXSPI\_IpTxFifoWatermarkEmptyFlag*** IP TX FIFO WaterMark empty.

***kFLEXSPI\_IpRxFifoWatermarkAvailableFlag*** IP RX FIFO WaterMark available.

***kFLEXSPI\_AhbCommandSequenceErrorFlag*** AHB triggered Command Sequences Error.

***kFLEXSPI\_IpCommandSequenceErrorFlag*** IP triggered Command Sequences Error.

***kFLEXSPI\_AhbCommandGrantTimeoutFlag*** AHB triggered Command Sequences Grant Timeout.

***kFLEXSPI\_IpCommandGrantTimeoutFlag*** IP triggered Command Sequences Grant Timeout.

***kFLEXSPI\_IpCommandExecutionDoneFlag*** IP triggered Command Sequences Execution finished.

***kFLEXSPI\_AllInterruptFlags*** All flags.

### 17.5.5 enum flexspi\_read\_sample\_clock\_t

Enumerator

***kFLEXSPI\_ReadSampleClkLoopbackInternally*** Dummy Read strobe generated by FlexSPI Controller and loopback internally.

***kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad*** Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

***kFLEXSPI\_ReadSampleClkLoopbackFromSckPad*** SCK output clock and loopback from SCK pad.

***kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad*** Flash provided Read strobe and input from DQS pad.

### 17.5.6 enum flexspi\_cs\_interval\_cycle\_unit\_t

Enumerator

***kFLEXSPI\_CsIntervalUnit1SckCycle*** Chip selection interval: CSINTERVAL \* 1 serial clock cycle.

***kFLEXSPI\_CsIntervalUnit256SckCycle*** Chip selection interval: CSINTERVAL \* 256 serial clock cycle.

### 17.5.7 enum flexspi\_ahb\_write\_wait\_unit\_t

Enumerator

***kFLEXSPI\_AhbWriteWaitUnit2AhbCycle*** AWRWAIT unit is 2 ahb clock cycle.

***kFLEXSPI\_AhbWriteWaitUnit8AhbCycle*** AWRWAIT unit is 8 ahb clock cycle.  
***kFLEXSPI\_AhbWriteWaitUnit32AhbCycle*** AWRWAIT unit is 32 ahb clock cycle.  
***kFLEXSPI\_AhbWriteWaitUnit128AhbCycle*** AWRWAIT unit is 128 ahb clock cycle.  
***kFLEXSPI\_AhbWriteWaitUnit512AhbCycle*** AWRWAIT unit is 512 ahb clock cycle.  
***kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle*** AWRWAIT unit is 2048 ahb clock cycle.  
***kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle*** AWRWAIT unit is 8192 ahb clock cycle.  
***kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle*** AWRWAIT unit is 32768 ahb clock cycle.

### 17.5.8 enum flexspi\_ip\_error\_code\_t

Enumerator

***kFLEXSPI\_IpCmdErrorNoError*** No error.  
***kFLEXSPI\_IpCmdErrorJumpOnCsInIpCmd*** IP command with JMP\_ON\_CS instruction used.  
***kFLEXSPI\_IpCmdErrorUnknownOpCode*** Unknown instruction opcode in the sequence.  
***kFLEXSPI\_IpCmdErrorSdrDummyInDdrSequence*** Instruction DUMMY\_SDR/DUMMY\_RW-DS\_SDR used in DDR sequence.  
***kFLEXSPI\_IpCmdErrorDdrDummyInSdrSequence*** Instruction DUMMY\_DDR/DUMMY\_RW-DS\_DDR used in SDR sequence.  
***kFLEXSPI\_IpCmdErrorInvalidAddress*** Flash access start address exceed the whole flash address range (A1/A2/B1/B2).  
***kFLEXSPI\_IpCmdErrorSequenceExecutionTimeout*** Sequence execution timeout.  
***kFLEXSPI\_IpCmdErrorFlashBoundaryAcrosss*** Flash boundary crossed.

### 17.5.9 enum flexspi\_ahb\_error\_code\_t

Enumerator

***kFLEXSPI\_AhbCmdErrorNoError*** No error.  
***kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd*** AHB Write command with JMP\_ON\_CS instruction used in the sequence.  
***kFLEXSPI\_AhbCmdErrorUnknownOpCode*** Unknown instruction opcode in the sequence.  
***kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence*** Instruction DUMMY\_SDR/DUMMY\_R-WDS\_SDR used in DDR sequence.  
***kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence*** Instruction DUMMY\_DDR/DUMMY\_R-WDS\_DDR used in SDR sequence.  
***kFLEXSPI\_AhbCmdSequenceExecutionTimeout*** Sequence execution timeout.

### 17.5.10 enum flexspi\_port\_t

Enumerator

***kFLEXSPI\_PortA1*** Access flash on A1 port.

- kFLEXSPI\_PortA2* Access flash on A2 port.
- kFLEXSPI\_PortB1* Access flash on B1 port.
- kFLEXSPI\_PortB2* Access flash on B2 port.

### 17.5.11 enum flexspi\_arb\_command\_source\_t

### 17.5.12 enum flexspi\_command\_type\_t

Enumerator

- kFLEXSPI\_Command* FlexSPI operation: Only command, both TX and Rx buffer are ignored.
- kFLEXSPI\_Config* FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

## 17.6 Function Documentation

### 17.6.1 uint32\_t FLEXSPI\_GetInstance ( **FLEXSPI\_Type** \* *base* )

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXSPI base pointer. |
|-------------|-----------------------|

### 17.6.2 status\_t FLEXSPI\_CheckAndClearError ( **FLEXSPI\_Type** \* *base*, **uint32\_t** *status* )

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXSPI base pointer. |
| <i>status</i> | interrupt status.     |

### 17.6.3 void FLEXSPI\_Init ( **FLEXSPI\_Type** \* *base*, **const flexspi\_config\_t** \* *config* )

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | FLEXSPI configure structure.     |

#### 17.6.4 void FLEXSPI\_GetDefaultConfig ( **flexspi\_config\_t** \* *config* )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>config</i> | FLEXSPI configuration structure. |
|---------------|----------------------------------|

#### 17.6.5 void FLEXSPI\_Deinit ( **FLEXSPI\_Type** \* *base* )

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

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

#### 17.6.6 void FLEXSPI\_UpdateDIIValue ( **FLEXSPI\_Type** \* *base*, **flexspi\_device\_config\_t** \* *config*, **flexspi\_port\_t** *port* )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | Flash configuration parameters.  |
| <i>port</i>   | FLEXSPI Operation port.          |

#### 17.6.7 void FLEXSPI\_SetFlashConfig ( **FLEXSPI\_Type** \* *base*, **flexspi\_device\_config\_t** \* *config*, **flexspi\_port\_t** *port* )

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | Flash configuration parameters.  |
| <i>port</i>   | FLEXSPI Operation port.          |

### 17.6.8 static void FLEXSPI\_SoftwareReset ( FLEXSPI\_Type \* *base* ) [inline], [static]

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

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

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

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                |
| <i>enable</i> | True means enable FLEXSPI, false means disable. |

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

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

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

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

**17.6.12 static void FLEXSPI\_EnableTxDMA ( FLEXSPI\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                               |
| <i>enable</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |

**17.6.13 static void FLEXSPI\_EnableRxDMA ( FLEXSPI\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                              |
| <i>enable</i> | Enable flag for receive DMA request. Pass true for enable, false for disable. |

**17.6.14 static uint32\_t FLEXSPI\_GetTxFifoAddress ( FLEXSPI\_Type \* *base* )  
[inline], [static]**

Parameters

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

Return values

|            |                  |
|------------|------------------|
| <i>The</i> | tx fifo address. |
|------------|------------------|

**17.6.15 static uint32\_t FLEXSPI\_GetRxFifoAddress ( FLEXSPI\_Type \* *base* )  
[inline], [static]**

Parameters

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

Return values

|            |                  |
|------------|------------------|
| <i>The</i> | rx fifo address. |
|------------|------------------|

### 17.6.16 static void FLEXSPI\_ResetFifos ( FLEXSPI\_Type \* *base*, bool *txFifo*, bool *rxFifo* ) [inline], [static]

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>txFifo</i> | Pass true to reset TX FIFO.      |
| <i>rxFifo</i> | Pass true to reset RX FIFO.      |

### 17.6.17 static void FLEXSPI\_GetFifoCounts ( FLEXSPI\_Type \* *base*, size\_t \* *txCount*, size\_t \* *rxCount* ) [inline], [static]

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | FLEXSPI peripheral base address.                                                                                             |
| out | <i>txCount</i> | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.  |

### 17.6.18 static uint32\_t FLEXSPI\_GetInterruptStatusFlags ( FLEXSPI\_Type \* *base* ) [inline], [static]

Parameters

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

Return values

|                  |                                                                                                |
|------------------|------------------------------------------------------------------------------------------------|
| <i>interrupt</i> | status flag, use status flag to AND <code>flexspi_flags_t</code> could get the related status. |
|------------------|------------------------------------------------------------------------------------------------|

### 17.6.19 static void FLEXSPI\_ClearInterruptStatusFlags ( `FLEXSPI_Type` \* *base*, `uint32_t` *mask* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

### 17.6.20 static `flexspi_arb_command_source_t` FLEXSPI\_GetArbitrator-CommandSource ( `FLEXSPI_Type` \* *base* ) [inline], [static]

Parameters

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

Return values

|                |                                     |
|----------------|-------------------------------------|
| <i>trigger</i> | source of current command sequence. |
|----------------|-------------------------------------|

### 17.6.21 static `flexspi_ip_error_code_t` FLEXSPI\_GetIPCommandErrorCode ( `FLEXSPI_Type` \* *base*, `uint8_t` \* *index* ) [inline], [static]

Parameters

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

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |
|--------------|---------------------------------------------------------------------------------------|

Return values

|              |                                      |
|--------------|--------------------------------------|
| <i>error</i> | code when IP command error detected. |
|--------------|--------------------------------------|

### 17.6.22 static flexspi\_ahb\_error\_code\_t FLEXSPI\_GetAHBCommandErrorCode ( FLEXSPI\_Type \* *base*, uint8\_t \* *index* ) [inline], [static]

Parameters

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                      |
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

|              |                                       |
|--------------|---------------------------------------|
| <i>error</i> | code when AHB command error detected. |
|--------------|---------------------------------------|

### 17.6.23 static bool FLEXSPI\_GetBusIdleStatus ( FLEXSPI\_Type \* *base* ) [inline], [static]

Parameters

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

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is idle. |
| <i>false</i> | Bus is busy. |

### 17.6.24 void FLEXSPI\_UpdateRxSampleClock ( FLEXSPI\_Type \* *base*, flexspi\_read\_sample\_clock\_t *clockSource* )

Parameters

|                    |                                                                 |
|--------------------|-----------------------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address.                                |
| <i>clockSource</i> | clockSource of type <a href="#">flexspi_read_sample_clock_t</a> |

#### 17.6.25 static void FLEXSPI\_EnableIPParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

#### 17.6.26 static void FLEXSPI\_EnableAHBParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

#### 17.6.27 void FLEXSPI\_UpdateLUT ( FLEXSPI\_Type \* *base*, uint32\_t *index*, const uint32\_t \* *cmd*, uint32\_t *count* )

Parameters

|              |                                                                                                                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                                                                                                                                                       |
| <i>index</i> | From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory. |

|              |                         |
|--------------|-------------------------|
| <i>cmd</i>   | Command sequence array. |
| <i>count</i> | Number of sequences.    |

**17.6.28 static void FLEXSPI\_WriteData ( FLEXSPI\_Type \* *base*, uint32\_t *data*, uint8\_t *fifoIndex* ) [inline], [static]**

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address |
| <i>data</i>      | The data bytes to send          |
| <i>fifoIndex</i> | Destination fifo index.         |

**17.6.29 static uint32\_t FLEXSPI\_ReadData ( FLEXSPI\_Type \* *base*, uint8\_t *fifoIndex* ) [inline], [static]**

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address |
| <i>fifoIndex</i> | Source fifo index.              |

Returns

The data in the FIFO.

**17.6.30 status\_t FLEXSPI\_WriteBlocking ( FLEXSPI\_Type \* *base*, uint8\_t \* *buffer*, size\_t *size* )**

Note

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

Parameters

---

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address  |
| <i>buffer</i> | The data bytes to send           |
| <i>size</i>   | The number of data bytes to send |

Return values

|                                                 |                                    |
|-------------------------------------------------|------------------------------------|
| <i>kStatus_Success</i>                          | write success without error        |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout         |
| <i>kStatus_FLEXSPI_IpCommandSequenceError</i>   | IP command sequence error detected |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected  |

### 17.6.31 status\_t FLEXSPI\_ReadBlocking ( **FLEXSPI\_Type** \* *base*, **uint8\_t** \* *buffer*, **size\_t** *size* )

Note

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

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address     |
| <i>buffer</i> | The data bytes to send              |
| <i>size</i>   | The number of data bytes to receive |

Return values

|                                                 |                            |
|-------------------------------------------------|----------------------------|
| <i>kStatus_Success</i>                          | read success without error |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout |

|                                                |                                    |
|------------------------------------------------|------------------------------------|
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i> | IP command sequence error detected |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>  | IP command grant timeout detected  |

### 17.6.32 **status\_t FLEXSPI\_TransferBlocking ( FLEXSPI\_Type \* *base*, flexspi\_transfer\_t \* *xfer* )**

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | FLEXSPI peripheral base address    |
| <i>xfer</i> | pointer to the transfer structure. |

Return values

|                                                 |                                        |
|-------------------------------------------------|----------------------------------------|
| <i>kStatus_Success</i>                          | command transfer success without error |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout             |
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i>  | IP command sequence error detected     |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>   | IP command grant timeout detected      |

### 17.6.33 **void FLEXSPI\_TransferCreateHandle ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle*, flexspi\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                   |
| <i>handle</i> | pointer to flexspi_handle_t structure to store the transfer state. |

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

### 17.6.34 status\_t FLEXSPI\_TransferNonBlocking ( **FLEXSPI\_Type** \* *base*, **flexspi\_handle\_t** \* *handle*, **flexspi\_transfer\_t** \* *xfer* )

Note

Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI\_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus\_FLEXSPI\_Busy, the transfer is finished. For FLEXSPI\_Read, the dataSize should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                              |
| <i>handle</i> | pointer to <b>flexspi_handle_t</b> structure which stores the transfer state. |
| <i>xfer</i>   | pointer to <b>flexspi_transfer_t</b> structure.                               |

Return values

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <i>kStatus_Success</i>      | Successfully start the data transmission. |
| <i>kStatus_FLEXSPI_Busy</i> | Previous transmission still not finished. |

### 17.6.35 status\_t FLEXSPI\_TransferGetCount ( **FLEXSPI\_Type** \* *base*, **flexspi\_handle\_t** \* *handle*, **size\_t** \* *count* )

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                              |
| <i>handle</i> | pointer to <b>flexspi_handle_t</b> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.           |

Return values

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

### 17.6.36 void FLEXSPI\_TransferAbort ( **FLEXSPI\_Type** \* *base*, **flexspi\_handle\_t** \* *handle* )

Note

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

Parameters

|               |                                                                              |
|---------------|------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                             |
| <i>handle</i> | pointer to <b>flexspi_handle_t</b> structure which stores the transfer state |

### 17.6.37 void FLEXSPI\_TransferHandleIRQ ( **FLEXSPI\_Type** \* *base*, **flexspi\_handle\_t** \* *handle* )

Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.              |
| <i>handle</i> | pointer to <b>flexspi_handle_t</b> structure. |

# Chapter 18

## GPT: General Purpose Timer

### 18.1 Overview

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

### 18.2 Function groups

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

#### 18.2.1 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.

### 18.3 Typical use case

#### 18.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gpt

## Data Structures

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

## Enumerations

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

*List of clock sources.*

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

## Driver version

- #define `FSL_GPT_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 4)`)

## Initialization and deinitialization

- void `GPT_Init` (GPT\_Type \*base, const `gpt_config_t` \*initConfig)
   
*Initialize GPT to reset state and initialize running mode.*

- void **GPT\_Deinit** (GPT\_Type \*base)  
*Disables the module and gates the GPT clock.*
- void **GPT\_GetDefaultConfig** (gpt\_config\_t \*config)  
*Fills in the GPT configuration structure with default settings.*

## Software Reset

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

## Clock source and frequency control

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

## Timer Start and Stop

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

## Read the timer period

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

## GPT Input/Output Signal Control

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

- static `gpt_output_operation_mode_t GPT_GetOutputOperationMode` (`GPT_Type *base, gpt_output_compare_channel_t channel`)
 

*Set GPT operation mode of output compare channel.*
- static `void GPT_SetOutputCompareValue` (`GPT_Type *base, gpt_output_compare_channel_t channel, uint32_t value`)
 

*Get GPT operation mode of output compare channel.*
- static `uint32_t GPT_GetOutputCompareValue` (`GPT_Type *base, gpt_output_compare_channel_t channel`)
 

*Set GPT output compare value of output compare channel.*
- static `void GPT_ForceOutput` (`GPT_Type *base, gpt_output_compare_channel_t channel`)
 

*Get GPT output compare value of output compare channel.*
- static void `GPT_ForceOutput` (`GPT_Type *base, gpt_output_compare_channel_t channel`)
 

*Force GPT output action on output compare channel, ignoring comparator.*

## GPT Interrupt and Status Interface

- static void `GPT_EnableInterrupts` (`GPT_Type *base, uint32_t mask`)
 

*Enables the selected GPT interrupts.*
- static void `GPT_DisableInterrupts` (`GPT_Type *base, uint32_t mask`)
 

*Disables the selected GPT interrupts.*
- static `uint32_t GPT_GetEnabledInterrupts` (`GPT_Type *base`)
 

*Gets the enabled GPT interrupts.*

## Status Interface

- static `uint32_t GPT_GetStatusFlags` (`GPT_Type *base, gpt_status_flag_t flags`)
 

*Get GPT status flags.*
- static void `GPT_ClearStatusFlags` (`GPT_Type *base, gpt_status_flag_t flags`)
 

*Clears the GPT status flags.*

## 18.4 Data Structure Documentation

### 18.4.1 struct gpt\_config\_t

#### Data Fields

- `gpt_clock_source_t clockSource`

*clock source for GPT module.*
- `uint32_t divider`

*clock divider (prescaler+1) from clock source to counter.*
- `bool enableFreeRun`

*true: FreeRun mode, false: Restart mode.*
- `bool enableRunInWait`

*GPT enabled in wait mode.*
- `bool enableRunInStop`

*GPT enabled in stop mode.*
- `bool enableRunInDoze`

*GPT enabled in doze mode.*
- `bool enableRunInDbg`

*GPT enabled in debug mode.*

- bool `enableMode`

*true:* counter reset to 0 when enabled;  
*false:* counter retain its value when enabled.

**Field Documentation**

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

**18.5 Enumeration Type Documentation****18.5.1 enum gpt\_clock\_source\_t**

## Note

Actual number of clock sources is SoC dependent

## Enumerator

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

**18.5.2 enum gpt\_input\_capture\_channel\_t**

## Enumerator

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

### 18.5.3 enum gpt\_input\_operation\_mode\_t

Enumerator

*kGPT\_InputOperation\_Disabled* Don't capture.

*kGPT\_InputOperation\_RiseEdge* Capture on rising edge of input pin.

*kGPT\_InputOperation\_FallEdge* Capture on falling edge of input pin.

*kGPT\_InputOperation\_BothEdge* Capture on both edges of input pin.

### 18.5.4 enum gpt\_output\_compare\_channel\_t

Enumerator

*kGPT\_OutputCompare\_Channel1* Output Compare Channel1.

*kGPT\_OutputCompare\_Channel2* Output Compare Channel2.

*kGPT\_OutputCompare\_Channel3* Output Compare Channel3.

### 18.5.5 enum gpt\_output\_operation\_mode\_t

Enumerator

*kGPT\_OutputOperation\_Disconnected* Don't change output pin.

*kGPT\_OutputOperation\_Toggle* Toggle output pin.

*kGPT\_OutputOperation\_Clear* Set output pin low.

*kGPT\_OutputOperation\_Set* Set output pin high.

*kGPT\_OutputOperation\_Activelow* Generate a active low pulse on output pin.

### 18.5.6 enum gpt\_interrupt\_enable\_t

Enumerator

*kGPT\_OutputCompare1InterruptEnable* Output Compare Channel1 interrupt enable.

*kGPT\_OutputCompare2InterruptEnable* Output Compare Channel2 interrupt enable.

*kGPT\_OutputCompare3InterruptEnable* Output Compare Channel3 interrupt enable.

*kGPT\_InputCapture1InterruptEnable* Input Capture Channel1 interrupt enable.

*kGPT\_InputCapture2InterruptEnable* Input Capture Channell1 interrupt enable.

*kGPT\_RollOverFlagInterruptEnable* Counter rolled over interrupt enable.

### 18.5.7 enum gpt\_status\_flag\_t

Enumerator

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

## 18.6 Function Documentation

### 18.6.1 void GPT\_Init ( *GPT\_Type* \* *base*, *const gpt\_config\_t* \* *initConfig* )

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | GPT peripheral base address.    |
| <i>initConfig</i> | GPT mode setting configuration. |

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

Parameters

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

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

The default values are:

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

Parameters

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

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

Parameters

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

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

Parameters

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

#### 18.6.6 static **gpt\_clock\_source\_t** GPT\_GetClockSource ( **GPT\_Type** \* *base* ) [inline], [static]

Parameters

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

Returns

clock source (see [gpt\\_clock\\_source\\_t](#) typedef enumeration).

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

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | Divider of GPT (1-4096).     |

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

Parameters

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

Returns

clock divider in GPT module (1-4096).

#### 18.6.9 static void GPT\_SetOscClockDivider ( **GPT\_Type** \* *base*, uint32\_t *divider* ) [inline], [static]

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | OSC Divider(1-16).           |

#### 18.6.10 static uint32\_t GPT\_GetOscClockDivider ( **GPT\_Type** \* *base* ) [inline], [static]

Parameters

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

Returns

OSC clock divider in GPT module (1-16).

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

Parameters

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

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

Parameters

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

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

Parameters

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

Returns

Current GPT counter value.

### 18.6.14 static void GPT\_SetInputOperationMode ( **GPT\_Type** \* *base*, **gpt\_input\_capture\_channel\_t** *channel*, **gpt\_input\_operation\_mode\_t** *mode* ) [inline], [static]

Parameters

|                |                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                           |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration).             |
| <i>mode</i>    | GPT input capture operation mode (see <a href="#">gpt_input_operation_mode_t</a> typedef enumeration). |

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

Parameters

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

Returns

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

### 18.6.16 static uint32\_t GPT\_GetInputCaptureValue ( **GPT\_Type** \* *base*, **gpt\_input\_capture\_channel\_t** *channel* ) [inline], [static]

Parameters

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

Returns

GPT input capture value.

### 18.6.17 static void GPT\_SetOutputOperationMode ( **GPT\_Type** \* *base*, **gpt\_output\_compare\_channel\_t** *channel*, **gpt\_output\_operation\_mode\_t** *mode* ) [inline], [static]

Parameters

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

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

Parameters

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

Returns

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

**18.6.19 static void GPT\_SetOutputCompareValue ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel*, uint32\_t *value* ) [inline], [static]**

Parameters

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

**18.6.20 static uint32\_t GPT\_GetOutputCompareValue ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel* ) [inline], [static]**

Parameters

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

Returns

GPT output compare value.

**18.6.21 static void GPT\_ForceOutput ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel* ) [inline], [static]**

Parameters

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

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

Parameters

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

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

Parameters

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

#### 18.6.24 static *uint32\_t* GPT\_GetEnabledInterrupts ( *GPT\_Type* \* *base* ) [[inline](#)], [[static](#)]

Parameters

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

Returns

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

#### 18.6.25 static *uint32\_t* GPT\_GetStatusFlags ( *GPT\_Type* \* *base*, *gpt\_status\_flag\_t* *flags* ) [[inline](#)], [[static](#)]

Parameters

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

Returns

GPT status, each bit represents one status flag.

### **18.6.26 static void GPT\_ClearStatusFlags ( GPT\_Type \* *base*, gpt\_status\_flag\_t *flags* ) [inline], [static]**

Parameters

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

# Chapter 19

## I3C: I3C Driver

### 19.1 Overview

#### Modules

- I3C Common Driver
- I3C Master Driver
- I3C Slave Driver

#### Data Structures

- struct `i3c_device_info_t`  
*I3C device information.* [More...](#)

#### Macros

- `#define I3C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`  
*Timeout times for waiting flag.*

#### Enumerations

- enum {  
    `kStatus_I3C_Busy` = MAKE\_STATUS(kStatusGroup\_I3C, 0),  
    `kStatus_I3C_Idle` = MAKE\_STATUS(kStatusGroup\_I3C, 1),  
    `kStatus_I3C_Nak` = MAKE\_STATUS(kStatusGroup\_I3C, 2),  
    `kStatus_I3C_WriteAbort`,  
    `kStatus_I3C_Term` = MAKE\_STATUS(kStatusGroup\_I3C, 4),  
    `kStatus_I3C_HdrParityError` = MAKE\_STATUS(kStatusGroup\_I3C, 5),  
    `kStatus_I3C_CrcError` = MAKE\_STATUS(kStatusGroup\_I3C, 6),  
    `kStatus_I3C_ReadFifoError` = MAKE\_STATUS(kStatusGroup\_I3C, 7),  
    `kStatus_I3C_WriteFifoError` = MAKE\_STATUS(kStatusGroup\_I3C, 8),  
    `kStatus_I3C_MsgError`,  
    `kStatus_I3C_InvalidReq` = MAKE\_STATUS(kStatusGroup\_I3C, 10),  
    `kStatus_I3C_Timeout` = MAKE\_STATUS(kStatusGroup\_I3C, 11),  
    `kStatus_I3C_SlaveCountExceed`,  
    `kStatus_I3C_IBIWon`,  
    `kStatus_I3C_OVERRUNError` = MAKE\_STATUS(kStatusGroup\_I3C, 14),  
    `kStatus_I3C_UnderrunError` = MAKE\_STATUS(kStatusGroup\_I3C, 15),  
    `kStatus_I3C_UnderrunNak`,  
    `kStatus_I3C_InvalidStart` = MAKE\_STATUS(kStatusGroup\_I3C, 17),  
    `kStatus_I3C_SdrParityError` = MAKE\_STATUS(kStatusGroup\_I3C, 18),  
    `kStatus_I3C_S0S1Error` = MAKE\_STATUS(kStatusGroup\_I3C, 19) }

- *I3C status return codes.*
- enum [i3c\\_hdr\\_mode\\_t](#)  
*I3C HDR modes.*

## Driver version

- #define [FSL\\_I3C\\_DRIVER\\_VERSION](#)([MAKE\\_VERSION](#)(2, 10, 7))  
*I3C driver version.*

## 19.2 Data Structure Documentation

### 19.2.1 struct i3c\_device\_info\_t

#### Data Fields

- uint8\_t [dynamicAddr](#)  
*Device dynamic address.*
- uint8\_t [staticAddr](#)  
*Static address.*
- uint8\_t [dcr](#)  
*Device characteristics register information.*
- uint8\_t [bcr](#)  
*Bus characteristics register information.*
- uint16\_t [vendorID](#)  
*Device vendor ID(manufacture ID).*
- uint32\_t [partNumber](#)  
*Device part number info.*
- uint16\_t [maxReadLength](#)  
*Maximum read length.*
- uint16\_t [maxWriteLength](#)  
*Maximum write length.*
- uint8\_t [hdrMode](#)  
*Support hdr mode, could be OR logic in i3c\_hdr\_mode.*

**Field Documentation**

- (1) `uint8_t i3c_device_info_t::dynamicAddr`
- (2) `uint8_t i3c_device_info_t::staticAddr`
- (3) `uint8_t i3c_device_info_t::dcr`
- (4) `uint8_t i3c_device_info_t::bcr`
- (5) `uint16_t i3c_device_info_t::vendorID`
- (6) `uint16_t i3c_device_info_t::maxReadLength`
- (7) `uint16_t i3c_device_info_t::maxWriteLength`
- (8) `uint8_t i3c_device_info_t::hdrMode`

**19.3 Macro Definition Documentation**

**19.3.1 #define I3C\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

**19.4 Enumeration Type Documentation****19.4.1 anonymous enum**

Enumerator

***kStatus\_I3C\_Busy*** The master is already performing a transfer.

***kStatus\_I3C\_Idle*** The slave driver is idle.

***kStatus\_I3C\_Nak*** The slave device sent a NAK in response to an address.

***kStatus\_I3C\_WriteAbort*** The slave device sent a NAK in response to a write.

***kStatus\_I3C\_Term*** The master terminates slave read.

***kStatus\_I3C\_HdrParityError*** Parity error from DDR read.

***kStatus\_I3C\_CrcError*** CRC error from DDR read.

***kStatus\_I3C\_ReadFifoError*** Read from M/SRDATA register when FIFO empty.

***kStatus\_I3C\_WriteFifoError*** Write to M/SWDATA register when FIFO full.

***kStatus\_I3C\_MsgError*** Message SDR/DDR mismatch or read/write message in wrong state.

***kStatus\_I3C\_InvalidReq*** Invalid use of request.

***kStatus\_I3C\_Timeout*** The module has stalled too long in a frame.

***kStatus\_I3C\_SlaveCountExceed*** The I3C slave count has exceed the definition in I3C\_MAX\_DEV\_CNT.

***kStatus\_I3C\_IBIWon*** The I3C slave event IBI or MR or HJ won the arbitration on a header address.

***kStatus\_I3C\_OverrunError*** Slave internal from-bus buffer/FIFO overrun.

***kStatus\_I3C\_UnderrunError*** Slave internal to-bus buffer/FIFO underrun.

***kStatus\_I3C\_UnderrunNak*** Slave internal from-bus buffer/FIFO underrun and NACK error.

*kStatus\_I3C\_InvalidStart* Slave invalid start flag.

*kStatus\_I3C\_SdrParityError* SDR parity error.

*kStatus\_I3C\_S0S1Error* S0 or S1 error.

#### 19.4.2 enum i3c\_hdr\_mode\_t

## 19.5 I3C Master Driver

### 19.5.1 Overview

#### Data Structures

- struct `i3c_register_ibl_addr_t`  
*Structure with setting master IBI rules and slave registry.* [More...](#)
- struct `i3c_baudrate_hz_t`  
*Structure with I3C baudrate settings.* [More...](#)
- struct `i3c_master_daa_baudrate_t`  
*I3C DAA baud rate configuration.* [More...](#)
- struct `i3c_master_config_t`  
*Structure with settings to initialize the I3C master module.* [More...](#)
- struct `i3c_master_transfer_callback_t`  
*i3c master callback functions.* [More...](#)
- struct `i3c_master_transfer_t`  
*Non-blocking transfer descriptor structure.* [More...](#)
- struct `i3c_master_handle_t`  
*Driver handle for master non-blocking APIs.* [More...](#)

#### Typedefs

- typedef void(\* `i3c_master_isr_t`)(I3C\_Type \*base, void \*handle)  
*Typedef for master interrupt handler.*

#### Enumerations

- enum `_i3c_master_flags` {
   
  `kI3C_MasterBetweenFlag` = I3C\_MSTATUS\_BETWEEN\_MASK,  
`kI3C_MasterNackDetectFlag` = I3C\_MSTATUS\_NACKED\_MASK,  
`kI3C_MasterSlaveStartFlag` = I3C\_MSTATUS\_SLVSTART\_MASK,  
`kI3C_MasterControlDoneFlag` = I3C\_MSTATUS\_MCTRLDONE\_MASK,  
`kI3C_MasterCompleteFlag` = I3C\_MSTATUS\_COMPLETE\_MASK,  
`kI3C_MasterRxReadyFlag` = I3C\_MSTATUS\_RXPEND\_MASK,  
`kI3C_MasterTxReadyFlag` = I3C\_MSTATUS\_TXNOTFULL\_MASK,  
`kI3C_MasterArbitrationWonFlag` = I3C\_MSTATUS\_IBIWON\_MASK,  
`kI3C_MasterErrorFlag` = I3C\_MSTATUS\_ERRWARN\_MASK,  
`kI3C_MasterSlave2MasterFlag` = I3C\_MSTATUS\_NOWMASTER\_MASK }
   
*I3C master peripheral flags.*
- enum `_i3c_master_error_flags` {

```

kI3C_MasterErrorNackFlag = I3C_MERRWARN_NACK_MASK,
kI3C_MasterErrorWriteAbortFlag = I3C_MERRWARN_WRABT_MASK,
kI3C_MasterErrorParityFlag = I3C_MERRWARN_HPAR_MASK,
kI3C_MasterErrorCrcFlag = I3C_MERRWARN_HCRC_MASK,
kI3C_MasterErrorReadFlag = I3C_MERRWARN_OREAD_MASK,
kI3C_MasterErrorWriteFlag = I3C_MERRWARN_OWRITE_MASK,
kI3C_MasterErrorMsgFlag = I3C_MERRWARN_MSGERR_MASK,
kI3C_MasterErrorInvalidReqFlag = I3C_MERRWARN_INVREQ_MASK,
kI3C_MasterErrorTimeoutFlag = I3C_MERRWARN_TIMEOUT_MASK,
kI3C_MasterAllErrorFlags }

```

*I3C master error flags to indicate the causes.*

- enum `i3c_master_state_t` {
   
kI3C\_MasterStateIdle = 0U,
   
kI3C\_MasterStateSlvReq = 1U,
   
kI3C\_MasterStateMsgSdr = 2U,
   
kI3C\_MasterStateNormAct = 3U,
   
kI3C\_MasterStateDdr = 4U,
   
kI3C\_MasterStateDaa = 5U,
   
kI3C\_MasterStateIbiAck = 6U,
   
kI3C\_MasterStateIbiRcv = 7U }

*I3C working master state.*

- enum `i3c_master_enable_t` {
   
kI3C\_MasterOff = 0U,
   
kI3C\_MasterOn = 1U,
   
kI3C\_MasterCapable = 2U }
- enum `i3c_master_hkeep_t` {
   
kI3C\_MasterHighKeeperNone = 0U,
   
kI3C\_MasterHighKeeperWiredIn = 1U,
   
kI3C\_MasterPassiveSDA = 2U,
   
kI3C\_MasterPassiveSDASCL = 3U }

*I3C high keeper configuration.*

- enum `i3c_bus_request_t` {
   
kI3C\_RequestNone = 0U,
   
kI3C\_RequestEmitStartAddr = 1U,
   
kI3C\_RequestEmitStop = 2U,
   
kI3C\_RequestIbiAckNack = 3U,
   
kI3C\_RequestProcessDAA = 4U,
   
kI3C\_RequestForceExit = 6U,
   
kI3C\_RequestAutoIbi = 7U }

*Emits the requested operation when doing in pieces vs.*

- enum `i3c_bus_type_t` {
   
kI3C\_TypeI3CSdr = 0U,
   
kI3C\_TypeI2C = 1U,
   
kI3C\_TypeI3CDdr = 2U }

*Bus type with EmitStartAddr:*

- enum `i3c_ibl_response_t` {
   
  `kI3C_IbiRespAck` = 0U,
   
  `kI3C_IbiRespNack` = 1U,
   
  `kI3C_IbiRespAckMandatory` = 2U,
   
  `kI3C_IbiRespManual` = 3U }

*IBI response.*

- enum `i3c_ibl_type_t` {
   
  `kI3C_IbiNormal` = 0U,
   
  `kI3C_IbiHotJoin` = 1U,
   
  `kI3C_IbiMasterRequest` = 2U }

*IBI type.*

- enum `i3c_ibl_state_t` {
   
  `kI3C_IbiReady` = 0U,
   
  `kI3C_IbiDataBuffNeed` = 1U,
   
  `kI3C_IbiAckNackPending` = 2U }

*IBI state.*

- enum `i3c_direction_t` {
   
  `kI3C_Write` = 0U,
   
  `kI3C_Read` = 1U }

*Direction of master and slave transfers.*

- enum `i3c_tx_trigger_level_t` {
   
  `kI3C_TxTriggerOnEmpty` = 0U,
   
  `kI3C_TxTriggerUntilOneQuarterOrLess` = 1U,
   
  `kI3C_TxTriggerUntilOneHalfOrLess` = 2U,
   
  `kI3C_TxTriggerUntilOneLessThanFull` = 3U }

*Watermark of TX int/dma trigger level.*

- enum `i3c_rx_trigger_level_t` {
   
  `kI3C_RxTriggerOnNotEmpty` = 0U,
   
  `kI3C_RxTriggerUntilOneQuarterOrMore` = 1U,
   
  `kI3C_RxTriggerUntilOneHalfOrMore` = 2U,
   
  `kI3C_RxTriggerUntilThreeQuarterOrMore` = 3U }

*Watermark of RX int/dma trigger level.*

- enum `i3c_rx_term_ops_t` {
   
  `kI3C_RxTermDisable` = 0U,
   
  `kI3C_RxAutoTerm` = 1U,
   
  `kI3C_RxTermLastByte` = 2U }

*I3C master read termination operations.*

- enum `_i3c_master_transfer_flags` {
   
  `kI3C_TransferDefaultFlag` = 0x00U,
   
  `kI3C_TransferNoStartFlag` = 0x01U,
   
  `kI3C_TransferRepeatedStartFlag` = 0x02U,
   
  `kI3C_TransferNoStopFlag` = 0x04U,
   
  `kI3C_TransferWordsFlag` = 0x08U,
   
  `kI3C_TransferDisableRxTermFlag` = 0x10U,
   
  `kI3C_TransferRxAutoTermFlag`,
   
  `kI3C_TransferStartWithBroadcastAddr` = 0x40U }

*Transfer option flags.*

## Initialization and deinitialization

- void **I3C\_MasterGetDefaultConfig** (*i3c\_master\_config\_t* \*masterConfig)  
*Provides a default configuration for the I3C master peripheral.*
- void **I3C\_MasterInit** (I3C\_Type \*base, const *i3c\_master\_config\_t* \*masterConfig, uint32\_t sourceClock\_Hz)  
*Initializes the I3C master peripheral.*
- void **I3C\_MasterDeinit** (I3C\_Type \*base)  
*Deinitializes the I3C master peripheral.*
- **status\_t I3C\_MasterCheckAndClearError** (I3C\_Type \*base, uint32\_t status)
- **status\_t I3C\_MasterWaitForCtrlDone** (I3C\_Type \*base, bool waitIdle)
- **status\_t I3C\_CheckForBusyBus** (I3C\_Type \*base)
- static void **I3C\_MasterEnable** (I3C\_Type \*base, *i3c\_master\_enable\_t* enable)  
*Set I3C module master mode.*
- void **I3C\_SlaveGetDefaultConfig** (*i3c\_slave\_config\_t* \*slaveConfig)  
*Provides a default configuration for the I3C slave peripheral.*
- void **I3C\_SlaveInit** (I3C\_Type \*base, const *i3c\_slave\_config\_t* \*slaveConfig, uint32\_t slowClock\_Hz)  
*Initializes the I3C slave peripheral.*
- void **I3C\_SlaveDeinit** (I3C\_Type \*base)  
*Deinitializes the I3C slave peripheral.*
- static void **I3C\_SlaveEnable** (I3C\_Type \*base, bool isEnabled)  
*Enable/Disable Slave.*

## Status

- static uint32\_t **I3C\_MasterGetStatusFlags** (I3C\_Type \*base)  
*Gets the I3C master status flags.*
- static void **I3C\_MasterClearStatusFlags** (I3C\_Type \*base, uint32\_t statusMask)  
*Clears the I3C master status flag state.*
- static uint32\_t **I3C\_MasterGetErrorHandlerStatusFlags** (I3C\_Type \*base)  
*Gets the I3C master error status flags.*
- static void **I3C\_MasterClearErrorHandlerStatusFlags** (I3C\_Type \*base, uint32\_t statusMask)  
*Clears the I3C master error status flag state.*
- *i3c\_master\_state\_t* **I3C\_MasterGetState** (I3C\_Type \*base)  
*Gets the I3C master state.*
- static uint32\_t **I3C\_SlaveGetStatusFlags** (I3C\_Type \*base)  
*Gets the I3C slave status flags.*
- static void **I3C\_SlaveClearStatusFlags** (I3C\_Type \*base, uint32\_t statusMask)  
*Clears the I3C slave status flag state.*
- static uint32\_t **I3C\_SlaveGetErrorHandlerStatusFlags** (I3C\_Type \*base)  
*Gets the I3C slave error status flags.*
- static void **I3C\_SlaveClearErrorHandlerStatusFlags** (I3C\_Type \*base, uint32\_t statusMask)  
*Clears the I3C slave error status flag state.*
- *i3c\_slave\_activity\_state\_t* **I3C\_SlaveGetActivityState** (I3C\_Type \*base)  
*Gets the I3C slave state.*
- **status\_t I3C\_SlaveCheckAndClearError** (I3C\_Type \*base, uint32\_t status)

## Interrupts

- static void **I3C\_MasterEnableInterrupts** (I3C\_Type \*base, uint32\_t interruptMask)  
*Enables the I3C master interrupt requests.*
- static void **I3C\_MasterDisableInterrupts** (I3C\_Type \*base, uint32\_t interruptMask)  
*Disables the I3C master interrupt requests.*
- static uint32\_t **I3C\_MasterGetEnabledInterrupts** (I3C\_Type \*base)  
*Returns the set of currently enabled I3C master interrupt requests.*
- static uint32\_t **I3C\_MasterGetPendingInterrupts** (I3C\_Type \*base)  
*Returns the set of pending I3C master interrupt requests.*
- static void **I3C\_SlaveEnableInterrupts** (I3C\_Type \*base, uint32\_t interruptMask)  
*Enables the I3C slave interrupt requests.*
- static void **I3C\_SlaveDisableInterrupts** (I3C\_Type \*base, uint32\_t interruptMask)  
*Disables the I3C slave interrupt requests.*
- static uint32\_t **I3C\_SlaveGetEnabledInterrupts** (I3C\_Type \*base)  
*Returns the set of currently enabled I3C slave interrupt requests.*
- static uint32\_t **I3C\_SlaveGetPendingInterrupts** (I3C\_Type \*base)  
*Returns the set of pending I3C slave interrupt requests.*

## DMA control

- static void **I3C\_MasterEnableDMA** (I3C\_Type \*base, bool enableTx, bool enableRx, uint32\_t width)  
*Enables or disables I3C master DMA requests.*
- static uint32\_t **I3C\_MasterGetTxFifoAddress** (I3C\_Type \*base, uint32\_t width)  
*Gets I3C master transmit data register address for DMA transfer.*
- static uint32\_t **I3C\_MasterGetRxFifoAddress** (I3C\_Type \*base, uint32\_t width)  
*Gets I3C master receive data register address for DMA transfer.*
- static void **I3C\_SlaveEnableDMA** (I3C\_Type \*base, bool enableTx, bool enableRx, uint32\_t width)  
*Enables or disables I3C slave DMA requests.*
- static uint32\_t **I3C\_SlaveGetTxFifoAddress** (I3C\_Type \*base, uint32\_t width)  
*Gets I3C slave transmit data register address for DMA transfer.*
- static uint32\_t **I3C\_SlaveGetRxFifoAddress** (I3C\_Type \*base, uint32\_t width)  
*Gets I3C slave receive data register address for DMA transfer.*

## FIFO control

- static void **I3C\_MasterSetWatermarks** (I3C\_Type \*base, i3c\_tx\_trigger\_level\_t txLvl, i3c\_rx\_trigger\_level\_t rxLvl, bool flushTx, bool flushRx)  
*Sets the watermarks for I3C master FIFOs.*
- static void **I3C\_MasterGetFifoCounts** (I3C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)  
*Gets the current number of bytes in the I3C master FIFOs.*
- static void **I3C\_SlaveSetWatermarks** (I3C\_Type \*base, i3c\_tx\_trigger\_level\_t txLvl, i3c\_rx\_trigger\_level\_t rxLvl, bool flushTx, bool flushRx)  
*Sets the watermarks for I3C slave FIFOs.*
- static void **I3C\_SlaveGetFifoCounts** (I3C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)  
*Gets the current number of bytes in the I3C slave FIFOs.*

## Bus operations

- void **I3C\_MasterSetBaudRate** (I3C\_Type \*base, const i3c\_baudrate\_hz\_t \*baudRate\_Hz, uint32\_t sourceClock\_Hz)
 

*Sets the I3C bus frequency for master transactions.*
- static bool **I3C\_MasterGetBusIdleState** (I3C\_Type \*base)
 

*Returns whether the bus is idle.*
- status\_t **I3C\_MasterStartWithRxSize** (I3C\_Type \*base, i3c\_bus\_type\_t type, uint8\_t address, i3c\_direction\_t dir, uint8\_t rxSize)
 

*Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.*
- status\_t **I3C\_MasterStart** (I3C\_Type \*base, i3c\_bus\_type\_t type, uint8\_t address, i3c\_direction\_t dir)
 

*Sends a START signal and slave address on the I2C/I3C bus.*
- status\_t **I3C\_MasterRepeatedStartWithRxSize** (I3C\_Type \*base, i3c\_bus\_type\_t type, uint8\_t address, i3c\_direction\_t dir, uint8\_t rxSize)
 

*Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.*
- static status\_t **I3C\_MasterRepeatedStart** (I3C\_Type \*base, i3c\_bus\_type\_t type, uint8\_t address, i3c\_direction\_t dir)
 

*Sends a repeated START signal and slave address on the I2C/I3C bus.*
- status\_t **I3C\_MasterSend** (I3C\_Type \*base, const void \*txBuff, size\_t txSize, uint32\_t flags)
 

*Performs a polling send transfer on the I2C/I3C bus.*
- status\_t **I3C\_MasterReceive** (I3C\_Type \*base, void \*rxBuff, size\_t rxSize, uint32\_t flags)
 

*Performs a polling receive transfer on the I2C/I3C bus.*
- status\_t **I3C\_MasterStop** (I3C\_Type \*base)
 

*Sends a STOP signal on the I2C/I3C bus.*
- void **I3C\_MasterEmitRequest** (I3C\_Type \*base, i3c\_bus\_request\_t masterReq)
 

*I3C master emit request.*
- static void **I3C\_MasterEmitIBIResponse** (I3C\_Type \*base, i3c\_ibi\_response\_t ibiResponse)
 

*I3C master emit request.*
- void **I3C\_MasterRegisterIBI** (I3C\_Type \*base, i3c\_register\_ibit\_addr\_t \*ibiRule)
 

*I3C master register IBI rule.*
- void **I3C\_MasterGetIBIRules** (I3C\_Type \*base, i3c\_register\_ibit\_addr\_t \*ibiRule)
 

*I3C master get IBI rule.*
- i3c\_ibit\_type\_t **I3C\_GetIBIType** (I3C\_Type \*base)
 

*I3C master get IBI Type.*
- static uint8\_t **I3C\_GetIBIAddress** (I3C\_Type \*base)
 

*I3C master get IBI Address.*
- status\_t **I3C\_MasterProcessDAAWithSpecifiedBaudrate** (I3C\_Type \*base, uint8\_t \*addressList, uint32\_t count, i3c\_master\_daa\_baudrate\_t \*daaBaudRate)
 

*Performs a DAA in the i3c bus with specified temporary baud rate.*
- static status\_t **I3C\_MasterProcessDAA** (I3C\_Type \*base, uint8\_t \*addressList, uint32\_t count)
 

*Performs a DAA in the i3c bus.*
- i3c\_device\_info\_t \* **I3C\_MasterGetDeviceListAfterDAA** (I3C\_Type \*base, uint8\_t \*count)
 

*Get device information list after DAA process is done.*
- status\_t **I3C\_MasterTransferBlocking** (I3C\_Type \*base, i3c\_master\_transfer\_t \*transfer)
 

*Performs a master polling transfer on the I2C/I3C bus.*
- void **I3C\_SlaveRequestEvent** (I3C\_Type \*base, i3c\_slave\_event\_t event)
 

*I3C slave request event.*
- status\_t **I3C\_SlaveSend** (I3C\_Type \*base, const void \*txBuff, size\_t txSize)

- **status\_t I3C\_SlaveReceive** (I3C\_Type \*base, void \*rxBuff, size\_t rxSize)  
*Performs a polling receive transfer on the I3C bus.*

## Non-blocking

- **void I3C\_MasterTransferCreateHandle** (I3C\_Type \*base, i3c\_master\_handle\_t \*handle, const i3c\_master\_transfer\_callback\_t \*callback, void \*userData)  
*Creates a new handle for the I3C master non-blocking APIs.*
- **status\_t I3C\_MasterTransferNonBlocking** (I3C\_Type \*base, i3c\_master\_handle\_t \*handle, i3c\_master\_transfer\_t \*transfer)  
*Performs a non-blocking transaction on the I2C/I3C bus.*
- **status\_t I3C\_MasterTransferGetCount** (I3C\_Type \*base, i3c\_master\_handle\_t \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- **void I3C\_MasterTransferAbort** (I3C\_Type \*base, i3c\_master\_handle\_t \*handle)  
*Terminates a non-blocking I3C master transmission early.*

## IRQ handler

- **void I3C\_MasterTransferHandleIRQ** (I3C\_Type \*base, void \*intHandle)  
*Reusable routine to handle master interrupts.*

### 19.5.2 Data Structure Documentation

#### 19.5.2.1 struct i3c\_register\_ibl\_addr\_t

##### Data Fields

- **uint8\_t address [5]**  
*Address array for registry.*
- **bool iblHasPayload**  
*Whether the address array has mandatory IBI byte.*

##### Field Documentation

- (1) **uint8\_t i3c\_register\_ibl\_addr\_t::address[5]**
- (2) **bool i3c\_register\_ibl\_addr\_t::iblHasPayload**

#### 19.5.2.2 struct i3c\_baudrate\_hz\_t

##### Data Fields

- **uint32\_t i2cBaud**  
*Desired I2C baud rate in Hertz.*

- `uint32_t i3cPushPullBaud`  
*Desired I3C push-pull baud rate in Hertz.*
- `uint32_t i3cOpenDrainBaud`  
*Desired I3C open-drain baud rate in Hertz.*

**Field Documentation**

- (1) `uint32_t i3c_baudrate_hz_t::i2cBaud`
- (2) `uint32_t i3c_baudrate_hz_t::i3cPushPullBaud`
- (3) `uint32_t i3c_baudrate_hz_t::i3cOpenDrainBaud`

**19.5.2.3 struct i3c\_master\_daa\_baudrate\_t****Data Fields**

- `uint32_t sourceClock_Hz`  
*FCLK, function clock in Hertz.*
- `uint32_t i3cPushPullBaud`  
*Desired I3C push-pull baud rate in Hertz.*
- `uint32_t i3cOpenDrainBaud`  
*Desired I3C open-drain baud rate in Hertz.*

**Field Documentation**

- (1) `uint32_t i3c_master_daa_baudrate_t::sourceClock_Hz`
- (2) `uint32_t i3c_master_daa_baudrate_t::i3cPushPullBaud`
- (3) `uint32_t i3c_master_daa_baudrate_t::i3cOpenDrainBaud`

**19.5.2.4 struct i3c\_master\_config\_t**

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the `I3C_MasterGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

**Data Fields**

- `i3c_master_enable_t enableMaster`  
*Enable master mode.*
- `bool disableTimeout`  
*Whether to disable timeout to prevent the ERRWARN.*
- `i3c_master_hkeep_t hKeep`  
*High keeper mode setting.*
- `bool enableOpenDrainStop`  
*Whether to emit open-drain speed STOP.*

- bool `enableOpenDrainHigh`  
*Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.*
- `i3c_baudrate_hz_t baudRate_Hz`  
*Desired baud rate settings.*

### Field Documentation

- (1) `i3c_master_enable_t i3c_master_config_t::enableMaster`
- (2) `bool i3c_master_config_t::disableTimeout`
- (3) `i3c_master_hkeep_t i3c_master_config_t::hKeep`
- (4) `bool i3c_master_config_t::enableOpenDrainStop`
- (5) `bool i3c_master_config_t::enableOpenDrainHigh`
- (6) `i3c_baudrate_hz_t i3c_master_config_t::baudRate_Hz`

### 19.5.2.5 struct i3c\_master\_transfer\_callback\_t

#### Data Fields

- void(\* `slave2Master` )(I3C\_Type \*base, void \*userData)  
*Transfer complete callback.*
- void(\* `ibiCallback` )(I3C\_Type \*base, i3c\_master\_handle\_t \*handle, `i3c_ibi_type_t` ibiType, `i3c_ibi_state_t` ibiState)  
*IBI event callback.*
- void(\* `transferComplete` )(I3C\_Type \*base, i3c\_master\_handle\_t \*handle, `status_t` completionStatus, void \*userData)  
*Transfer complete callback.*

### 19.5.2.6 struct \_i3c\_master\_transfer

This structure is used to pass transaction parameters to the `I3C_MasterTransferNonBlocking()` API.

#### Data Fields

- `uint32_t flags`  
*Bit mask of options for the transfer.*
- `uint8_t slaveAddress`  
*The 7-bit slave address.*
- `i3c_direction_t direction`  
*Either `kI3C_Read` or `kI3C_Write`.*
- `uint32_t subaddress`  
*Sub address.*
- `size_t subaddressSize`  
*Length of sub address to send in bytes.*
- `void * data`

- `size_t dataSize`  
*Pointer to data to transfer.*
- `i3c_bus_type_t busType`  
*Number of bytes to transfer.*
- `i3c_ibi_response_t ibiResponse`  
*bus type.*
- `i3c_ibi_response_t ibiResponse`  
*ibi response during transfer.*

## Field Documentation

### (1) `uint32_t i3c_master_transfer_t::flags`

See enumeration `_i3c_master_transfer_flags` for available options. Set to 0 or `kI3C_TransferDefaultFlag` for normal transfers.

### (2) `uint8_t i3c_master_transfer_t::slaveAddress`

### (3) `i3c_direction_t i3c_master_transfer_t::direction`

### (4) `uint32_t i3c_master_transfer_t::subaddress`

Transferred MSB first.

### (5) `size_t i3c_master_transfer_t::subaddressSize`

Maximum size is 4 bytes.

### (6) `void* i3c_master_transfer_t::data`

### (7) `size_t i3c_master_transfer_t::dataSize`

### (8) `i3c_bus_type_t i3c_master_transfer_t::busType`

### (9) `i3c_ibi_response_t i3c_master_transfer_t::ibiResponse`

## 19.5.2.7 `struct _i3c_master_handle`

Note

The contents of this structure are private and subject to change.

## Data Fields

- `uint8_t state`  
*Transfer state machine current state.*
- `uint32_t remainingBytes`  
*Remaining byte count in current state.*
- `i3c_rx_term_ops_t rxTermOps`  
*Read termination operation.*
- `i3c_master_transfer_t transfer`  
*Copy of the current transfer info.*

- `uint8_t ibiAddress`  
*Slave address which request IBI.*
- `uint8_t * ibiBuff`  
*Pointer to IBI buffer to keep ibi bytes.*
- `size_t ibiPayloadSize`  
*IBI payload size.*
- `i3c_ibit_type_t ibiType`  
*IBI type.*
- `i3c_master_transfer_callback_t callback`  
*Callback functions pointer.*
- `void * userData`  
*Application data passed to callback.*

## Field Documentation

- (1) `uint8_t i3c_master_handle_t::state`
- (2) `uint32_t i3c_master_handle_t::remainingBytes`
- (3) `i3c_rx_term_ops_t i3c_master_handle_t::rxTermOps`
- (4) `i3c_master_transfer_t i3c_master_handle_t::transfer`
- (5) `uint8_t i3c_master_handle_t::ibiAddress`
- (6) `uint8_t* i3c_master_handle_t::ibiBuff`
- (7) `size_t i3c_master_handle_t::ibiPayloadSize`
- (8) `i3c_ibit_type_t i3c_master_handle_t::ibiType`
- (9) `i3c_master_transfer_callback_t i3c_master_handle_t::callback`
- (10) `void* i3c_master_handle_t::userData`

## 19.5.3 Typedef Documentation

### 19.5.3.1 `typedef void(* i3c_master_isr_t)(I3C_Type *base, void *handle)`

## 19.5.4 Enumeration Type Documentation

### 19.5.4.1 `enum _i3c_master_flags`

The following status register flags can be cleared:

- `kI3C_MasterSlaveStartFlag`
- `kI3C_MasterControlDoneFlag`
- `kI3C_MasterCompleteFlag`
- `kI3C_MasterArbitrationWonFlag`

- [kI3C\\_MasterSlave2MasterFlag](#)

All flags except [kI3C\\_MasterBetweenFlag](#) and [kI3C\\_MasterNackDetectFlag](#) can be enabled as interrupts.

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

- kI3C\_MasterBetweenFlag** Between messages/DAAs flag.
- kI3C\_MasterNackDetectFlag** NACK detected flag.
- kI3C\_MasterSlaveStartFlag** Slave request start flag.
- kI3C\_MasterControlDoneFlag** Master request complete flag.
- kI3C\_MasterCompleteFlag** Transfer complete flag.
- kI3C\_MasterRxReadyFlag** Rx data ready in Rx buffer flag.
- kI3C\_MasterTxReadyFlag** Tx buffer ready for Tx data flag.
- kI3C\_MasterArbitrationWonFlag** Header address won arbitration flag.
- kI3C\_MasterErrorFlag** Error occurred flag.
- kI3C\_MasterSlave2MasterFlag** Switch from slave to master flag.

#### 19.5.4.2 enum \_i3c\_master\_error\_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

- kI3C\_MasterErrorNackFlag** Slave NACKed the last address.
- kI3C\_MasterErrorWriteAbortFlag** Slave NACKed the write data.
- kI3C\_MasterErrorParityFlag** Parity error from DDR read.
- kI3C\_MasterErrorCrcFlag** CRC error from DDR read.
- kI3C\_MasterErrorReadFlag** Read from MRDATAB register when FIFO empty.
- kI3C\_MasterErrorWriteFlag** Write to MWDATAB register when FIFO full.
- kI3C\_MasterErrorMsgFlag** Message SDR/DDR mismatch or read/write message in wrong state.
- kI3C\_MasterErrorInvalidReqFlag** Invalid use of request.
- kI3C\_MasterErrorTimeoutFlag** The module has stalled too long in a frame.
- kI3C\_MasterAllErrorFlags** All error flags.

#### 19.5.4.3 enum i3c\_master\_state\_t

Enumerator

- kI3C\_MasterStateIdle** Bus stopped.

- kI3C\_MasterStateSlvReq* Bus stopped but slave holding SDA low.
- kI3C\_MasterStateMsgSdr* In SDR Message mode from using MWMSG\_SDR.
- kI3C\_MasterStateNormAct* In normal active SDR mode.
- kI3C\_MasterStateDdr* In DDR Message mode.
- kI3C\_MasterStateDaa* In ENTDAA mode.
- kI3C\_MasterStateIbiAck* Waiting on IBI ACK/NACK decision.
- kI3C\_MasterStateIbiRcv* receiving IBI.

#### 19.5.4.4 enum i3c\_master\_enable\_t

Enumerator

- kI3C\_MasterOff* Master off.
- kI3C\_MasterOn* Master on.
- kI3C\_MasterCapable* Master capable.

#### 19.5.4.5 enum i3c\_master\_hkeep\_t

Enumerator

- kI3C\_MasterHighKeeperNone* Use PUR to hold SCL high.
- kI3C\_MasterHighKeeperWiredIn* Use pin\_HK controls.
- kI3C\_MasterPassiveSDA* Hi-Z for Bus Free and hold SDA.
- kI3C\_MasterPassiveSDASCL* Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

#### 19.5.4.6 enum i3c\_bus\_request\_t

by message.

Enumerator

- kI3C\_RequestNone* No request.
- kI3C\_RequestEmitStartAddr* Request to emit start and address on bus.
- kI3C\_RequestEmitStop* Request to emit stop on bus.
- kI3C\_RequestIbiAckNack* Manual IBI ACK or NACK.
- kI3C\_RequestProcessDAA* Process DAA.
- kI3C\_RequestForceExit* Request to force exit.
- kI3C\_RequestAutoIbi* Hold in stopped state, but Auto-emit START,7E.

#### 19.5.4.7 enum i3c\_bus\_type\_t

Enumerator

*kI3C\_TypeI3CSdr* SDR mode of I3C.

*kI3C\_TypeI2C* Standard i2c protocol.

*kI3C\_TypeI3CDdr* HDR-DDR mode of I3C.

#### 19.5.4.8 enum i3c\_ibl\_response\_t

Enumerator

*kI3C\_IbiRespAck* ACK with no mandatory byte.

*kI3C\_IbiRespNack* NACK.

*kI3C\_IbiRespAckMandatory* ACK with mandatory byte.

*kI3C\_IbiRespManual* Reserved.

#### 19.5.4.9 enum i3c\_ibl\_type\_t

Enumerator

*kI3C\_IbiNormal* In-band interrupt.

*kI3C\_IbiHotJoin* slave hot join.

*kI3C\_IbiMasterRequest* slave master ship request.

#### 19.5.4.10 enum i3c\_ibl\_state\_t

Enumerator

*kI3C\_IbiReady* In-band interrupt ready state, ready for user to handle.

*kI3C\_IbiDataBuffNeed* In-band interrupt need data buffer for data receive.

*kI3C\_IbiAckNackPending* In-band interrupt Ack/Nack pending for decision.

#### 19.5.4.11 enum i3c\_direction\_t

Enumerator

*kI3C\_Write* Master transmit.

*kI3C\_Read* Master receive.

#### 19.5.4.12 enum i3c\_tx\_trigger\_level\_t

Enumerator

- kI3C\_TxTriggerOnEmpty* Trigger on empty.
- kI3C\_TxTriggerUntilOneQuarterOrLess* Trigger on 1/4 full or less.
- kI3C\_TxTriggerUntilOneHalfOrLess* Trigger on 1/2 full or less.
- kI3C\_TxTriggerUntilOneLessThanFull* Trigger on 1 less than full or less.

#### 19.5.4.13 enum i3c\_rx\_trigger\_level\_t

Enumerator

- kI3C\_RxTriggerOnNotEmpty* Trigger on not empty.
- kI3C\_RxTriggerUntilOneQuarterOrMore* Trigger on 1/4 full or more.
- kI3C\_RxTriggerUntilOneHalfOrMore* Trigger on 1/2 full or more.
- kI3C\_RxTriggerUntilThreeQuarterOrMore* Trigger on 3/4 full or more.

#### 19.5.4.14 enum i3c\_rx\_term\_ops\_t

Enumerator

- kI3C\_RxTermDisable* Master doesn't terminate read, used for CCC transfer.
- kI3C\_RxAutoTerm* Master auto terminate read after receiving specified bytes(<=255).
- kI3C\_RxTermLastByte* Master terminates read at any time after START, no length limitation.

#### 19.5.4.15 enum \_i3c\_master\_transfer\_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the [\\_i3c\\_master\\_transfer::flags](#) field.

Enumerator

- kI3C\_TransferDefaultFlag* Transfer starts with a start signal, stops with a stop signal.
- kI3C\_TransferNoStartFlag* Don't send a start condition, address, and sub address.
- kI3C\_TransferRepeatedStartFlag* Send a repeated start condition.
- kI3C\_TransferNoStopFlag* Don't send a stop condition.
- kI3C\_TransferWordsFlag* Transfer in words, else transfer in bytes.
- kI3C\_TransferDisableRxTermFlag* Disable Rx termination. Note: It's for I3C CCC transfer.
- kI3C\_TransferRxAutoTermFlag* Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C\_MasterReceive.
- kI3C\_TransferStartWithBroadcastAddr* Start transfer with 0x7E, then read/write data with device address.

## 19.5.5 Function Documentation

### 19.5.5.1 void I3C\_MasterGetDefaultConfig ( i3c\_master\_config\_t \* *masterConfig* )

This function provides the following default configuration for the I3C master peripheral:

```
* masterConfig->enableMaster = kI3C_MasterOn;
* masterConfig->disableTimeout = false;
* masterConfig->hKeep = kI3C_MasterHighKeeperNone;
* masterConfig->enableOpenDrainStop = true;
* masterConfig->enableOpenDrainHigh = true;
* masterConfig->baudRate_Hz = 100000U;
* masterConfig->busType = kI3C_TypeI2C;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with [I3C\\_MasterInit\(\)](#).

Parameters

|     |                     |                                                                                                          |
|-----|---------------------|----------------------------------------------------------------------------------------------------------|
| out | <i>masterConfig</i> | User provided configuration structure for default values. Refer to <a href="#">i3c_master_config_t</a> . |
|-----|---------------------|----------------------------------------------------------------------------------------------------------|

### 19.5.5.2 void I3C\_MasterInit ( I3C\_Type \* *base*, const i3c\_master\_config\_t \* *masterConfig*, uint32\_t *sourceClock\_Hz* )

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

|                       |                                                                                                                                          |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The I3C peripheral base address.                                                                                                         |
| <i>masterConfig</i>   | User provided peripheral configuration. Use <a href="#">I3C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.            |

### 19.5.5.3 void I3C\_MasterDeinit ( I3C\_Type \* *base* )

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

#### 19.5.5.4 static void I3C\_MasterEnable ( I3C\_Type \* *base*, i3c\_master\_enable\_t *enable* ) [inline], [static]

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | The I3C peripheral base address. |
| <i>enable</i> | Enable master mode.              |

#### 19.5.5.5 static uint32\_t I3C\_MasterGetStatusFlags ( I3C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_i3c\\_master\\_flags](#)

#### 19.5.5.6 static void I3C\_MasterClearStatusFlags ( I3C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- kI3C\_MasterSlaveStartFlag
- kI3C\_MasterControlDoneFlag
- kI3C\_MasterCompleteFlag
- kI3C\_MasterArbitrationWonFlag
- kI3C\_MasterSlave2MasterFlag

Attempts to clear other flags has no effect.

## Parameters

|                   |                                                                                                                                                                                                                                   |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I3C peripheral base address.                                                                                                                                                                                                  |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i3c_master_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I3C_MasterGetStatusFlags()</a> . |

## See Also

[\\_i3c\\_master\\_flags](#).**19.5.5.7 static uint32\_t I3C\_MasterGetErrorStatusFlags( I3C\_Type \* *base* ) [inline], [static]**

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

## Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

## See Also

[\\_i3c\\_master\\_error\\_flags](#)**19.5.5.8 static void I3C\_MasterClearErrorStatusFlags( I3C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]**

## Parameters

|                   |                                                                                                                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I3C peripheral base address.                                                                                                                                                                                                              |
| <i>statusMask</i> | A bitmask of error status flags that are to be cleared. The mask is composed of <a href="#">_i3c-master_error_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I3C_MasterGetStatusFlags()</a> . |

See Also

[\\_i3c\\_master\\_error\\_flags](#).

#### 19.5.5.9 **i3c\_master\_state\_t I3C\_MasterGetState ( I3C\_Type \* *base* )**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

I3C master state.

#### 19.5.5.10 **static void I3C\_MasterEnableInterrupts ( I3C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

All flags except [kI3C\\_MasterBetweenFlag](#) and [kI3C\\_MasterNackDetectFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I3C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_i3c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 19.5.5.11 **static void I3C\_MasterDisableInterrupts ( I3C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

All flags except [kI3C\\_MasterBetweenFlag](#) and [kI3C\\_MasterNackDetectFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I3C peripheral base address.                                                                                                                     |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_i3c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 19.5.5.12 static uint32\_t I3C\_MasterGetEnabledInterrupts ( I3C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

A bitmask composed of [\\_i3c\\_master\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

#### 19.5.5.13 static uint32\_t I3C\_MasterGetPendingInterrupts ( I3C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

A bitmask composed of [\\_i3c\\_master\\_flags](#) enumerators OR'd together to indicate the set of pending interrupts.

#### 19.5.5.14 static void I3C\_MasterEnableDMA ( I3C\_Type \* *base*, bool *enableTx*, bool *enableRx*, uint32\_t *width* ) [inline], [static]

Parameters

---

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>base</i>     | The I3C peripheral base address.                                               |
| <i>enableTx</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |
| <i>enableRx</i> | Enable flag for receive DMA request. Pass true for enable, false for disable.  |
| <i>width</i>    | DMA read/write unit in bytes.                                                  |

**19.5.5.15 static uint32\_t I3C\_MasterGetTxFifoAddress ( I3C\_Type \* *base*, uint32\_t *width* ) [inline], [static]**

Parameters

|              |                                  |
|--------------|----------------------------------|
| <i>base</i>  | The I3C peripheral base address. |
| <i>width</i> | DMA read/write unit in bytes.    |

Returns

The I3C Master Transmit Data Register address.

**19.5.5.16 static uint32\_t I3C\_MasterGetRxFifoAddress ( I3C\_Type \* *base*, uint32\_t *width* ) [inline], [static]**

Parameters

|              |                                  |
|--------------|----------------------------------|
| <i>base</i>  | The I3C peripheral base address. |
| <i>width</i> | DMA read/write unit in bytes.    |

Returns

The I3C Master Receive Data Register address.

**19.5.5.17 static void I3C\_MasterSetWatermarks ( I3C\_Type \* *base*, i3c\_tx\_trigger\_level\_t *txLvl*, i3c\_rx\_trigger\_level\_t *rxLvl*, bool *flushTx*, bool *flushRx* ) [inline], [static]**

Parameters

|                |                                                                                                                                                                |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                                                                                               |
| <i>txLvl</i>   | Transmit FIFO watermark level. The <a href="#">kI3C_MasterTxReadyFlag</a> flag is set whenever the number of words in the transmit FIFO reaches <i>txLvl</i> . |
| <i>rxLvl</i>   | Receive FIFO watermark level. The <a href="#">kI3C_MasterRxReadyFlag</a> flag is set whenever the number of words in the receive FIFO reaches <i>rxLvl</i> .   |
| <i>flushTx</i> | true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.                                                                                         |
| <i>flushRx</i> | true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.                                                                                         |

**19.5.5.18 static void I3C\_MasterGetFifoCounts ( I3C\_Type \* *base*, size\_t \* *rxCount*, size\_t \* *txCount* ) [inline], [static]**

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | The I3C peripheral base address.                                                                                             |
| out | <i>txCount</i> | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.  |

**19.5.5.19 void I3C\_MasterSetBaudRate ( I3C\_Type \* *base*, const i3c\_baudrate\_hz\_t \* *baudRate\_Hz*, uint32\_t *sourceClock\_Hz* )**

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <i>base</i>           | The I3C peripheral base address.                          |
| <i>baudRate_Hz</i>    | Pointer to structure of requested bus frequency in Hertz. |
| <i>sourceClock_Hz</i> | I3C functional clock frequency in Hertz.                  |

**19.5.5.20 static bool I3C\_MasterGetBusIdleState ( I3C\_Type \* *base* ) [inline], [static]**

Requires the master mode to be enabled.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

#### 19.5.5.21 status\_t I3C\_MasterStartWithRxSize ( I3C\_Type \* *base*, i3c\_bus\_type\_t *type*, uint8\_t *address*, i3c\_direction\_t *dir*, uint8\_t *rxSize* )

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

|                |                                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                                                                                                                |
| <i>type</i>    | The bus type to use in this transaction.                                                                                                                                        |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                      |
| <i>dir</i>     | Master transfer direction, either <a href="#">kI3C_Read</a> or <a href="#">kI3C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |
| <i>rxSize</i>  | Read terminate size for the followed read transfer, limit to 255 bytes.                                                                                                         |

Return values

|                                  |                                                                            |
|----------------------------------|----------------------------------------------------------------------------|
| <a href="#">kStatus_Success</a>  | START signal and address were successfully enqueued in the transmit FI-FO. |
| <a href="#">kStatus_I3C_Busy</a> | Another master is currently utilizing the bus.                             |

#### 19.5.5.22 status\_t I3C\_MasterStart ( I3C\_Type \* *base*, i3c\_bus\_type\_t *type*, uint8\_t *address*, i3c\_direction\_t *dir* )

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

## Parameters

|                |                                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                                                                                                                |
| <i>type</i>    | The bus type to use in this transaction.                                                                                                                                        |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                      |
| <i>dir</i>     | Master transfer direction, either <a href="#">kI3C_Read</a> or <a href="#">kI3C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

## Return values

|                                  |                                                                           |
|----------------------------------|---------------------------------------------------------------------------|
| <a href="#">kStatus_Success</a>  | START signal and address were successfully enqueued in the transmit FIFO. |
| <a href="#">kStatus_I3C_Busy</a> | Another master is currently utilizing the bus.                            |

**19.5.5.23 status\_t I3C\_MasterRepeatedStartWithRxSize ( I3C\_Type \* *base*, i3c\_bus\_type\_t *type*, uint8\_t *address*, i3c\_direction\_t *dir*, uint8\_t *rxSize* )**

This function is used to send a Repeated START signal when a transfer is already in progress. Like [I3C\\_MasterStart\(\)](#), it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the rxSize = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the rxSize configuration.

## Note

This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

## Parameters

|                |                                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                                                                                                                |
| <i>type</i>    | The bus type to use in this transaction.                                                                                                                                        |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                      |
| <i>dir</i>     | Master transfer direction, either <a href="#">kI3C_Read</a> or <a href="#">kI3C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>rxSize</i> | Read terminate size for the followed read transfer, limit to 255 bytes. |
|---------------|-------------------------------------------------------------------------|

Return values

|                        |                                                                                    |
|------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i> | Repeated START signal and address were successfully enqueued in the transmit FIFO. |
|------------------------|------------------------------------------------------------------------------------|

#### 19.5.5.24 static status\_t I3C\_MasterRepeatedStart ( *I3C\_Type* \* *base*, *i3c\_bus\_type\_t type*, *uint8\_t address*, *i3c\_direction\_t dir* ) [inline], [static]

This function is used to send a Repeated START signal when a transfer is already in progress. Like [I3C\\_MasterStart\(\)](#), it also sends the specified 7-bit address.

Note

This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

|                |                                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                                                                                                                |
| <i>type</i>    | The bus type to use in this transaction.                                                                                                                                        |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                      |
| <i>dir</i>     | Master transfer direction, either <a href="#">kI3C_Read</a> or <a href="#">kI3C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                        |                                                                                    |
|------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i> | Repeated START signal and address were successfully enqueued in the transmit FIFO. |
|------------------------|------------------------------------------------------------------------------------|

#### 19.5.5.25 status\_t I3C\_MasterSend ( *I3C\_Type* \* *base*, *const void* \* *txBuff*, *size\_t txSize*, *uint32\_t flags* )

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns [kStatus\\_I3C\\_Nak](#).

## Parameters

|               |                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I3C peripheral base address.                                                                                        |
| <i>txBuff</i> | The pointer to the data to be transferred.                                                                              |
| <i>txSize</i> | The length in bytes of the data to be transferred.                                                                      |
| <i>flags</i>  | Bit mask of options for the transfer. See enumeration <a href="#">_i3c_master_transfer_flags</a> for available options. |

## Return values

|                                             |                                                                |
|---------------------------------------------|----------------------------------------------------------------|
| <a href="#">kStatus_Success</a>             | Data was sent successfully.                                    |
| <a href="#">kStatus_I3C_Busy</a>            | Another master is currently utilizing the bus.                 |
| <a href="#">kStatus_I3C_Timeout</a>         | The module has stalled too long in a frame.                    |
| <a href="#">kStatus_I3C_Nak</a>             | The slave device sent a NAK in response to an address.         |
| <a href="#">kStatus_I3C_WriteAbort</a>      | The slave device sent a NAK in response to a write.            |
| <a href="#">kStatus_I3C_MsgError</a>        | Message SDR/DDR mismatch or read/write message in wrong state. |
| <a href="#">kStatus_I3C_WriteFifo-Error</a> | Write to M/SWDATA register when FIFO full.                     |
| <a href="#">kStatus_I3C_InvalidReq</a>      | Invalid use of request.                                        |

**19.5.5.26 status\_t I3C\_MasterReceive ( I3C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize*, uint32\_t *flags* )**

## Parameters

|               |                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I3C peripheral base address.                                                                                        |
| <i>rxBuff</i> | The pointer to the data to be transferred.                                                                              |
| <i>rxSize</i> | The length in bytes of the data to be transferred.                                                                      |
| <i>flags</i>  | Bit mask of options for the transfer. See enumeration <a href="#">_i3c_master_transfer_flags</a> for available options. |

## Return values

|                                 |                                 |
|---------------------------------|---------------------------------|
| <a href="#">kStatus_Success</a> | Data was received successfully. |
|---------------------------------|---------------------------------|

|                                    |                                                                |
|------------------------------------|----------------------------------------------------------------|
| <i>kStatus_I3C_Busy</i>            | Another master is currently utilizing the bus.                 |
| <i>kStatus_I3C_Timeout</i>         | The module has stalled too long in a frame.                    |
| <i>kStatus_I3C_Term</i>            | The master terminates slave read.                              |
| <i>kStatus_I3C_HdrParity-Error</i> | Parity error from DDR read.                                    |
| <i>kStatus_I3C_CrcError</i>        | CRC error from DDR read.                                       |
| <i>kStatus_I3C_MsgError</i>        | Message SDR/DDR mismatch or read/write message in wrong state. |
| <i>kStatus_I3C_ReadFifo-Error</i>  | Read from M/SRDATA register when FIFO empty.                   |
| <i>kStatus_I3C_InvalidReq</i>      | Invalid use of request.                                        |

#### 19.5.5.27 status\_t I3C\_MasterStop ( I3C\_Type \* *base* )

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Return values

|                               |                                                                                  |
|-------------------------------|----------------------------------------------------------------------------------|
| <i>kStatus_Success</i>        | The STOP signal was successfully sent on the bus and the transaction terminated. |
| <i>kStatus_I3C_Busy</i>       | Another master is currently utilizing the bus.                                   |
| <i>kStatus_I3C_Timeout</i>    | The module has stalled too long in a frame.                                      |
| <i>kStatus_I3C_InvalidReq</i> | Invalid use of request.                                                          |

#### 19.5.5.28 void I3C\_MasterEmitRequest ( I3C\_Type \* *base*, i3c\_bus\_request\_t *masterReq* )

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

|                  |                                                              |
|------------------|--------------------------------------------------------------|
| <i>masterReq</i> | I3C master request of type <a href="#">i3c_bus_request_t</a> |
|------------------|--------------------------------------------------------------|

**19.5.5.29 static void I3C\_MasterEmitIBIResponse ( *I3C\_Type* \* *base*, *i3c\_ibi\_response\_t* *ibiResponse* ) [inline], [static]**

Parameters

|                    |                                                                         |
|--------------------|-------------------------------------------------------------------------|
| <i>base</i>        | The I3C peripheral base address.                                        |
| <i>ibiResponse</i> | I3C master emit IBI response of type <a href="#">i3c_ibi_response_t</a> |

**19.5.5.30 void I3C\_MasterRegisterIBI ( *I3C\_Type* \* *base*, *i3c\_register\_ibit\_addr\_t* \* *ibiRule* )**

Parameters

|                |                                                                                  |
|----------------|----------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                 |
| <i>ibiRule</i> | Pointer to ibi rule description of type <a href="#">i3c_register_ibit_addr_t</a> |

**19.5.5.31 void I3C\_MasterGetIBIRules ( *I3C\_Type* \* *base*, *i3c\_register\_ibit\_addr\_t* \* *ibiRule* )**

Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                    |
| <i>ibiRule</i> | Pointer to store the read out ibi rule description. |

**19.5.5.32 *i3c\_ibit\_type\_t* I3C\_GetIBIType ( *I3C\_Type* \* *base* )**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Return values

|                       |                                          |
|-----------------------|------------------------------------------|
| <i>i3c_ibl_type_t</i> | Type of <a href="#">i3c_ibl_type_t</a> . |
|-----------------------|------------------------------------------|

### 19.5.5.33 static uint8\_t I3C\_GetIBIAddress ( I3C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Return values

|            |                    |
|------------|--------------------|
| <i>The</i> | 8-bit IBI address. |
|------------|--------------------|

### 19.5.5.34 status\_t I3C\_MasterProcessDAAWithSpecifiedBaudrate ( I3C\_Type \* *base*, uint8\_t \* *addressList*, uint32\_t *count*, i3c\_master\_daa\_baudrate\_t \* *daaBaudRate* )

Parameters

|                    |                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | The I3C peripheral base address.                                                                                                                                           |
| <i>addressList</i> | The pointer for address list which is used to do DAA.                                                                                                                      |
| <i>count</i>       | The address count in the address list.                                                                                                                                     |
| <i>daaBaudRate</i> | The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function. |

Return values

|                                               |                                                                                                             |
|-----------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <a href="#">kStatus_Success</a>               | The transaction was started successfully.                                                                   |
| <a href="#">kStatus_I3C_Busy</a>              | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |
| <a href="#">kStatus_I3C_SlaveCount_Exceed</a> | The I3C slave count has exceed the definition in I3C_MAX_DEV_CNT.                                           |

### 19.5.5.35 static status\_t I3C\_MasterProcessDAA ( I3C\_Type \* *base*, uint8\_t \* *addressList*, uint32\_t *count* ) [inline], [static]

Parameters

|                    |                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | The I3C peripheral base address.                                                                                                          |
| <i>addressList</i> | The pointer for address list which is used to do DAA.                                                                                     |
| <i>count</i>       | The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function. |

Return values

|                                      |                                                                                                             |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>               | The transaction was started successfully.                                                                   |
| <i>kStatus_I3C_Busy</i>              | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |
| <i>kStatus_I3C_SlaveCount_Exceed</i> | The I3C slave count has exceed the definition in I3C_MAX_DEV_CNT.                                           |

#### 19.5.5.36 `i3c_device_info_t* I3C_MasterGetDeviceListAfterDAA ( I3C_Type * base, uint8_t * count )`

Parameters

|            |              |                                                  |
|------------|--------------|--------------------------------------------------|
|            | <i>base</i>  | The I3C peripheral base address.                 |
| <i>out</i> | <i>count</i> | The pointer to store the available device count. |

Returns

Pointer to the `i3c_device_info_t` array.

#### 19.5.5.37 `status_t I3C_MasterTransferBlocking ( I3C_Type * base, i3c_master_transfer_t * transfer )`

Note

The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>base</i>     | The I3C peripheral base address.   |
| <i>transfer</i> | Pointer to the transfer structure. |

Return values

|                                    |                                                                              |
|------------------------------------|------------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | Data was received successfully.                                              |
| <i>kStatus_I3C_Busy</i>            | Another master is currently utilizing the bus.                               |
| <i>kStatus_I3C_IBIWon</i>          | The I3C slave event IBI or MR or HJ won the arbitration on a header address. |
| <i>kStatus_I3C_Timeout</i>         | The module has stalled too long in a frame.                                  |
| <i>kStatus_I3C_Nak</i>             | The slave device sent a NAK in response to an address.                       |
| <i>kStatus_I3C_WriteAbort</i>      | The slave device sent a NAK in response to a write.                          |
| <i>kStatus_I3C_Term</i>            | The master terminates slave read.                                            |
| <i>kStatus_I3C_HdrParity-Error</i> | Parity error from DDR read.                                                  |
| <i>kStatus_I3C_CrcError</i>        | CRC error from DDR read.                                                     |
| <i>kStatus_I3C_MsgError</i>        | Message SDR/DDR mismatch or read/write message in wrong state.               |
| <i>kStatus_I3C_ReadFifo-Error</i>  | Read from M/SRDATA register when FIFO empty.                                 |
| <i>kStatus_I3C_WriteFifo-Error</i> | Write to M/SWDATA register when FIFO full.                                   |
| <i>kStatus_I3C_InvalidReq</i>      | Invalid use of request.                                                      |

#### 19.5.5.38 **void I3C\_MasterTransferCreateHandle( I3C\_Type \* *base*, i3c\_master\_handle\_t \* *handle*, const i3c\_master\_transfer\_callback\_t \* *callback*, void \* *userData* )**

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I3C\\_MasterTransferAbort\(\)](#) API shall be called.

Note

The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The I3C peripheral base address.                             |
| out | <i>handle</i>   | Pointer to the I3C master driver handle.                     |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

#### 19.5.5.39 status\_t I3C\_MasterTransferNonBlocking ( *I3C\_Type \* base,* *i3c\_master\_handle\_t \* handle, i3c\_master\_transfer\_t \* transfer* )

Parameters

|  |                 |                                          |
|--|-----------------|------------------------------------------|
|  | <i>base</i>     | The I3C peripheral base address.         |
|  | <i>handle</i>   | Pointer to the I3C master driver handle. |
|  | <i>transfer</i> | The pointer to the transfer descriptor.  |

Return values

|                         |                                                                                                             |
|-------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>  | The transaction was started successfully.                                                                   |
| <i>kStatus_I3C_Busy</i> | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |

#### 19.5.5.40 status\_t I3C\_MasterTransferGetCount ( *I3C\_Type \* base, i3c\_master\_handle\_t* *\* handle, size\_t \* count* )

Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The I3C peripheral base address.                                    |
|     | <i>handle</i> | Pointer to the I3C master driver handle.                            |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

Return values

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

#### 19.5.5.41 void I3C\_MasterTransferAbort ( *I3C\_Type \* base, i3c\_master\_handle\_t \** *handle* )

## Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

## Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | The I3C peripheral base address.         |
| <i>handle</i> | Pointer to the I3C master driver handle. |

## Return values

|                         |                                                                |
|-------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>  | A transaction was successfully aborted.                        |
| <i>kStatus_I3C_Idle</i> | There is not a non-blocking transaction currently in progress. |

**19.5.5.42 void I3C\_MasterTransferHandleIRQ ( I3C\_Type \* *base*, void \* *intHandle* )**

## Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

## Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | The I3C peripheral base address.         |
| <i>intHandle</i> | Pointer to the I3C master driver handle. |

**19.5.5.43 void I3C\_SlaveGetDefaultConfig ( i3c\_slave\_config\_t \* *slaveConfig* )**

This function provides the following default configuration for the I3C slave peripheral:

```
* slaveConfig->enableslave = true;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with [I3C\\_SlaveInit\(\)](#).

## Parameters

|            |                    |                                                                                                         |
|------------|--------------------|---------------------------------------------------------------------------------------------------------|
| <i>out</i> | <i>slaveConfig</i> | User provided configuration structure for default values. Refer to <a href="#">i3c_slave_config_t</a> . |
|------------|--------------------|---------------------------------------------------------------------------------------------------------|

**19.5.5.44 void I3C\_SlaveInit ( I3C\_Type \* *base*, const i3c\_slave\_config\_t \* *slaveConfig*, uint32\_t *slowClock\_Hz* )**

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

|                     |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | The I3C peripheral base address.                                                                                                        |
| <i>slaveConfig</i>  | User provided peripheral configuration. Use <a href="#">I3C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>slowClock_Hz</i> | Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values.                                             |

#### 19.5.5.45 void I3C\_SlaveDeinit ( I3C\_Type \* *base* )

This function disables the I3C slave peripheral and gates the clock.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

#### 19.5.5.46 static void I3C\_SlaveEnable ( I3C\_Type \* *base*, bool *isEnabled* ) [inline], [static]

Parameters

|                  |                                  |
|------------------|----------------------------------|
| <i>base</i>      | The I3C peripheral base address. |
| <i>isEnabled</i> | Enable or disable.               |

#### 19.5.5.47 static uint32\_t I3C\_SlaveGetStatusFlags ( I3C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_i3c\\_slave\\_flags](#)

#### 19.5.5.48 static void I3C\_SlaveClearStatusFlags ( I3C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- kI3C\_SlaveBusStartFlag
- kI3C\_SlaveMatchedFlag
- kI3C\_SlaveBusStopFlag

Attempts to clear other flags has no effect.

Parameters

|                   |                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I3C peripheral base address.                                                                                                                                                                                                |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i3c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I3C_SlaveGetStatusFlags()</a> . |

See Also

[\\_i3c\\_slave\\_flags](#).

#### 19.5.5.49 static uint32\_t I3C\_SlaveGetErrorStatusFlags ( I3C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_i3c\\_slave\\_error\\_flags](#)

19.5.5.50 **static void I3C\_SlaveClearErrorStatusFlags ( I3C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]**

## Parameters

|                   |                                                                                                                                                                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I3C peripheral base address.                                                                                                                                                                                                                 |
| <i>statusMask</i> | A bitmask of error status flags that are to be cleared. The mask is composed of <a href="#">_i3c_slave_error_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I3C_SlaveGetErrorStatusFlags()</a> . |

## See Also

[\\_i3c\\_slave\\_error\\_flags](#).**19.5.5.51 i3c\_slave\_activity\_state\_t I3C\_SlaveGetActivityState ( I3C\_Type \* *base* )**

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

## Returns

I3C slave activity state, refer [i3c\\_slave\\_activity\\_state\\_t](#).**19.5.5.52 static void I3C\_SlaveEnableInterrupts ( I3C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

Only below flags can be enabled as interrupts.

- [kI3C\\_SlaveBusStartFlag](#)
- [kI3C\\_SlaveMatchedFlag](#)
- [kI3C\\_SlaveBusStopFlag](#)
- [kI3C\\_SlaveRxReadyFlag](#)
- [kI3C\\_SlaveTxReadyFlag](#)
- [kI3C\\_SlaveDynamicAddrChangedFlag](#)
- [kI3C\\_SlaveReceivedCCCFlag](#)
- [kI3C\\_SlaveErrorFlag](#)
- [kI3C\\_SlaveHDRCommandMatchFlag](#)
- [kI3C\\_SlaveCCCHandledFlag](#)
- [kI3C\\_SlaveEventSentFlag](#)

Parameters

|                      |                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I3C peripheral base address.                                                                                                                   |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_i3c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 19.5.5.53 static void I3C\_SlaveDisableInterrupts ( I3C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Only below flags can be disabled as interrupts.

- [kI3C\\_SlaveBusStartFlag](#)
- [kI3C\\_SlaveMatchedFlag](#)
- [kI3C\\_SlaveBusStopFlag](#)
- [kI3C\\_SlaveRxReadyFlag](#)
- [kI3C\\_SlaveTxReadyFlag](#)
- [kI3C\\_SlaveDynamicAddrChangedFlag](#)
- [kI3C\\_SlaveReceivedCCCFlag](#)
- [kI3C\\_SlaveErrorFlag](#)
- [kI3C\\_SlaveHDRCommandMatchFlag](#)
- [kI3C\\_SlaveCCCHandledFlag](#)
- [kI3C\\_SlaveEventSentFlag](#)

Parameters

|                      |                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I3C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_i3c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 19.5.5.54 static uint32\_t I3C\_SlaveGetEnabledInterrupts ( I3C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

A bitmask composed of [\\_i3c\\_slave\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

19.5.5.55 **static uint32\_t I3C\_SlaveGetPendingInterrupts ( I3C\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

A bitmask composed of [\\_i3c\\_slave\\_flags](#) enumerators OR'd together to indicate the set of pending interrupts.

#### 19.5.5.56 static void I3C\_SlaveEnableDMA ( I3C\_Type \* *base*, bool *enableTx*, bool *enableRx*, uint32\_t *width* ) [inline], [static]

Parameters

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>base</i>     | The I3C peripheral base address.                                               |
| <i>enableTx</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |
| <i>enableRx</i> | Enable flag for receive DMA request. Pass true for enable, false for disable.  |
| <i>width</i>    | DMA read/write unit in bytes.                                                  |

#### 19.5.5.57 static uint32\_t I3C\_SlaveGetTxFifoAddress ( I3C\_Type \* *base*, uint32\_t *width* ) [inline], [static]

Parameters

|              |                                  |
|--------------|----------------------------------|
| <i>base</i>  | The I3C peripheral base address. |
| <i>width</i> | DMA read/write unit in bytes.    |

Returns

The I3C Slave Transmit Data Register address.

#### 19.5.5.58 static uint32\_t I3C\_SlaveGetRxFifoAddress ( I3C\_Type \* *base*, uint32\_t *width* ) [inline], [static]

Parameters

|              |                                  |
|--------------|----------------------------------|
| <i>base</i>  | The I3C peripheral base address. |
| <i>width</i> | DMA read/write unit in bytes.    |

Returns

The I3C Slave Receive Data Register address.

**19.5.5.59 static void I3C\_SlaveSetWatermarks ( I3C\_Type \* *base*, i3c\_tx\_trigger\_level\_t *txLvl*, i3c\_rx\_trigger\_level\_t *rxLvl*, bool *flushTx*, bool *flushRx* ) [inline], [static]**

Parameters

|                |                                                                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                                                                                              |
| <i>txLvl</i>   | Transmit FIFO watermark level. The <a href="#">kI3C_SlaveTxReadyFlag</a> flag is set whenever the number of words in the transmit FIFO reaches <i>txLvl</i> . |
| <i>rxLvl</i>   | Receive FIFO watermark level. The <a href="#">kI3C_SlaveRxReadyFlag</a> flag is set whenever the number of words in the receive FIFO reaches <i>rxLvl</i> .   |
| <i>flushTx</i> | true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.                                                                                        |
| <i>flushRx</i> | true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.                                                                                        |

**19.5.5.60 static void I3C\_SlaveGetFifoCounts ( I3C\_Type \* *base*, size\_t \* *rxCount*, size\_t \* *txCount* ) [inline], [static]**

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | The I3C peripheral base address.                                                                                             |
| out | <i>txCount</i> | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.  |

**19.5.5.61 void I3C\_SlaveRequestEvent ( I3C\_Type \* *base*, i3c\_slave\_event\_t *event* )**

Parameters

|              |                                                           |
|--------------|-----------------------------------------------------------|
| <i>base</i>  | The I3C peripheral base address.                          |
| <i>event</i> | I3C slave event of type <a href="#">i3c_slave_event_t</a> |

#### 19.5.5.62 status\_t I3C\_SlaveSend ( I3C\_Type \* *base*, const void \* *txBuff*, size\_t *txSize* )

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I3C peripheral base address.                   |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

Returns

Error or success status returned by API.

#### 19.5.5.63 status\_t I3C\_SlaveReceive ( I3C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize* )

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I3C peripheral base address.                   |
| <i>rxBuff</i> | The pointer to the data to be transferred.         |
| <i>rxSize</i> | The length in bytes of the data to be transferred. |

Returns

Error or success status returned by API.

## 19.6 I3C Slave Driver

### 19.6.1 Overview

#### Data Structures

- struct `i3c_slave_config_t`  
*Structure with settings to initialize the I3C slave module.* [More...](#)
- struct `i3c_slave_transfer_t`  
*I3C slave transfer structure.* [More...](#)
- struct `i3c_slave_handle_t`  
*I3C slave handle structure.* [More...](#)

#### Typedefs

- typedef void(\* `i3c_slave_transfer_callback_t`)(I3C\_Type \*base, `i3c_slave_transfer_t` \*transfer, void \*userData)  
*Slave event callback function pointer type.*
- typedef void(\* `i3c_slave_isr_t`)(I3C\_Type \*base, void \*handle)  
*Typedef for slave interrupt handler.*

#### Enumerations

- enum `_i3c_slave_flags` {
   
`kI3C_SlaveNotStopFlag` = I3C\_SSTATUS\_STNOTSTOP\_MASK,  
`kI3C_SlaveMessageFlag` = I3C\_SSTATUS\_STMSG\_MASK,  
`kI3C_SlaveRequiredReadFlag` = I3C\_SSTATUS\_STREQRD\_MASK,  
`kI3C_SlaveRequiredWriteFlag` = I3C\_SSTATUS\_STREQWR\_MASK,  
`kI3C_SlaveBusDAAFlag` = I3C\_SSTATUS\_STDAA\_MASK,  
`kI3C_SlaveBusHDRModeFlag` = I3C\_SSTATUS\_STHDR\_MASK,  
`kI3C_SlaveBusStartFlag` = I3C\_SSTATUS\_START\_MASK,  
`kI3C_SlaveMatchedFlag` = I3C\_SSTATUS\_MATCHED\_MASK,  
`kI3C_SlaveBusStopFlag` = I3C\_SSTATUS\_STOP\_MASK,  
`kI3C_SlaveRxReadyFlag` = I3C\_SSTATUS\_RX\_PEND\_MASK,  
`kI3C_SlaveTxReadyFlag` = I3C\_SSTATUS\_TXNOTFULL\_MASK,  
`kI3C_SlaveDynamicAddrChangedFlag`,  
`kI3C_SlaveReceivedCCCFlag` = I3C\_SSTATUS\_CCC\_MASK,  
`kI3C_SlaveErrorFlag` = I3C\_SSTATUS\_ERRWARN\_MASK,  
`kI3C_SlaveHDRCommandMatchFlag` = I3C\_SSTATUS\_HDRMATCH\_MASK,  
`kI3C_SlaveCCCHandledFlag`,  
`kI3C_SlaveEventSentFlag` = I3C\_SSTATUS\_EVENT\_MASK,  
`kI3C_SlaveIbiDisableFlag` = I3C\_SSTATUS\_IBIDIS\_MASK,  
`kI3C_SlaveMasterRequestDisabledFlag` = I3C\_SSTATUS\_MRDIS\_MASK,  
`kI3C_SlaveHotJoinDisabledFlag` = I3C\_SSTATUS\_HJDIS\_MASK,  
`kI3C_SlaveClearFlags` = `kI3C_SlaveBusStartFlag` | `kI3C_SlaveMatchedFlag` | `kI3C_SlaveBusStopFlag`

- Flag }
- I3C slave peripheral flags.*
- enum `_i3c_slave_error_flags` {
   
kI3C\_SlaveErrorOverrunFlag = I3C\_SERRWARN\_ORUN\_MASK,  
 kI3C\_SlaveErrorUnderrunFlag = I3C\_SERRWARN\_URUN\_MASK,  
 kI3C\_SlaveErrorUnderrunNakFlag,  
 kI3C\_SlaveErrorTermFlag = I3C\_SERRWARN\_TERM\_MASK,  
 kI3C\_SlaveErrorInvalidStartFlag = I3C\_SERRWARN\_INVSTART\_MASK,  
 kI3C\_SlaveErrorSdrParityFlag = I3C\_SERRWARN\_SPAR\_MASK,  
 kI3C\_SlaveErrorHdrParityFlag = I3C\_SERRWARN\_HPAR\_MASK,  
 kI3C\_SlaveErrorHdrCRCFlag = I3C\_SERRWARN\_HCRC\_MASK,  
 kI3C\_SlaveErrorS0S1Flag = I3C\_SERRWARN\_S0S1\_MASK,  
 kI3C\_SlaveErrorOverreadFlag = I3C\_SERRWARN\_OREAD\_MASK,  
 kI3C\_SlaveErrorOverwriteFlag = I3C\_SERRWARN\_OWRITE\_MASK }
- I3C slave error flags to indicate the causes.*
- enum `i3c_slave_event_t` {
   
kI3C\_SlaveEventNormal = 0U,  
 kI3C\_SlaveEventIBI = 1U,  
 kI3C\_SlaveEventMasterReq = 2U,  
 kI3C\_SlaveEventHotJoinReq = 3U }
- I3C slave.event.*
- enum `i3c_slave_activity_state_t` {
   
kI3C\_SlaveNoLatency = 0U,  
 kI3C\_SlaveLatency1Ms = 1U,  
 kI3C\_SlaveLatency100Ms = 2U,  
 kI3C\_SlaveLatency10S = 3U }
- I3C slave.activity state.*
- enum `i3c_slave_transfer_event_t` {
   
kI3C\_SlaveAddressMatchEvent = 0x01U,  
 kI3C\_SlaveTransmitEvent = 0x02U,  
 kI3C\_SlaveReceiveEvent = 0x04U,  
 kI3C\_SlaveRequiredTransmitEvent = 0x08U,  
 kI3C\_SlaveStartEvent = 0x10U,  
 kI3C\_SlaveHDRCommandMatchEvent = 0x20U,  
 kI3C\_SlaveCompletionEvent = 0x40U,  
 kI3C\_SlaveRequestSentEvent = 0x80U,  
 kI3C\_SlaveReceivedCCCEvent = 0x100L,  
 kI3C\_SlaveAllEvents }
- Set of events sent to the callback for non blocking slave transfers.*

## Initialization and deinitialization

- void `I3C_SlaveGetDefaultConfig` (`i3c_slave_config_t` \*`slaveConfig`)
   
*Provides a default configuration for the I3C slave peripheral.*
- void `I3C_SlaveInit` (`I3C_Type` \*`base`, const `i3c_slave_config_t` \*`slaveConfig`, `uint32_t` `slowClock_-`

Hz)

*Initializes the I3C slave peripheral.*

- void **I3C\_SlaveDeinit** (I3C\_Type \*base)  
*Deinitializes the I3C slave peripheral.*
- static void **I3C\_SlaveEnable** (I3C\_Type \*base, bool isEnabled)  
*Enable/Disable Slave.*

## Status

- static uint32\_t **I3C\_SlaveGetStatusFlags** (I3C\_Type \*base)  
*Gets the I3C slave status flags.*
- static void **I3C\_SlaveClearStatusFlags** (I3C\_Type \*base, uint32\_t statusMask)  
*Clears the I3C slave status flag state.*
- static uint32\_t **I3C\_SlaveGetErrorStatusFlags** (I3C\_Type \*base)  
*Gets the I3C slave error status flags.*
- static void **I3C\_SlaveClearErrorStatusFlags** (I3C\_Type \*base, uint32\_t statusMask)  
*Clears the I3C slave error status flag state.*
- **i3c\_slave\_activity\_state\_t I3C\_SlaveGetActivityState** (I3C\_Type \*base)  
*Gets the I3C slave state.*
- **status\_t I3C\_SlaveCheckAndClearError** (I3C\_Type \*base, uint32\_t status)

## Interrupts

- static void **I3C\_SlaveEnableInterrupts** (I3C\_Type \*base, uint32\_t interruptMask)  
*Enables the I3C slave interrupt requests.*
- static void **I3C\_SlaveDisableInterrupts** (I3C\_Type \*base, uint32\_t interruptMask)  
*Disables the I3C slave interrupt requests.*
- static uint32\_t **I3C\_SlaveGetEnabledInterrupts** (I3C\_Type \*base)  
*Returns the set of currently enabled I3C slave interrupt requests.*
- static uint32\_t **I3C\_SlaveGetPendingInterrupts** (I3C\_Type \*base)  
*Returns the set of pending I3C slave interrupt requests.*

## DMA control

- static void **I3C\_SlaveEnableDMA** (I3C\_Type \*base, bool enableTx, bool enableRx, uint32\_t width)  
*Enables or disables I3C slave DMA requests.*
- static uint32\_t **I3C\_SlaveGetTxFifoAddress** (I3C\_Type \*base, uint32\_t width)  
*Gets I3C slave transmit data register address for DMA transfer.*
- static uint32\_t **I3C\_SlaveGetRxFifoAddress** (I3C\_Type \*base, uint32\_t width)  
*Gets I3C slave receive data register address for DMA transfer.*

## FIFO control

- static void **I3C\_SlaveSetWatermarks** (I3C\_Type \*base, **i3c\_tx\_trigger\_level\_t** txLvl, **i3c\_rx\_trigger\_level\_t** rxLvl, bool flushTx, bool flushRx)  
*Sets the watermarks for I3C slave FIFOs.*

- static void [I3C\\_SlaveGetFifoCounts](#) (I3C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)  
*Gets the current number of bytes in the I3C slave FIFOs.*

## Bus operations

- void [I3C\\_SlaveRequestEvent](#) (I3C\_Type \*base, i3c\_slave\_event\_t event)  
*I3C slave request event.*
- status\_t [I3C\\_SlaveSend](#) (I3C\_Type \*base, const void \*txBuff, size\_t txSize)  
*Performs a polling send transfer on the I3C bus.*
- status\_t [I3C\\_SlaveReceive](#) (I3C\_Type \*base, void \*rxBuff, size\_t rxSize)  
*Performs a polling receive transfer on the I3C bus.*

## Slave non-blocking

- void [I3C\\_SlaveTransferCreateHandle](#) (I3C\_Type \*base, i3c\_slave\_handle\_t \*handle, i3c\_slave\_transfer\_callback\_t callback, void \*userData)  
*Creates a new handle for the I3C slave non-blocking APIs.*
- status\_t [I3C\\_SlaveTransferNonBlocking](#) (I3C\_Type \*base, i3c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- status\_t [I3C\\_SlaveTransferGetCount](#) (I3C\_Type \*base, i3c\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer status during a non-blocking transfer.*
- void [I3C\\_SlaveTransferAbort](#) (I3C\_Type \*base, i3c\_slave\_handle\_t \*handle)  
*Aborts the slave non-blocking transfers.*

## Slave IRQ handler

- void [I3C\\_SlaveTransferHandleIRQ](#) (I3C\_Type \*base, void \*intHandle)  
*Reusable routine to handle slave interrupts.*
- void [I3C\\_SlaveRequestIBIWithData](#) (I3C\_Type \*base, uint8\_t \*data, size\_t dataSize)  
*I3C slave request IBI event with data payload(mandatory and extended).*
- void [I3C\\_SlaveRequestIBIWithSingleData](#) (I3C\_Type \*base, uint8\_t data, size\_t dataSize)  
*I3C slave request IBI event with single data.*

## 19.6.2 Data Structure Documentation

### 19.6.2.1 struct i3c\_slave\_config\_t

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the [I3C\\_SlaveGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## Data Fields

- bool `enableSlave`  
*Whether to enable slave.*
- bool `isHotJoin`  
*Whether to enable slave hotjoin before enable slave.*
- uint8\_t `staticAddr`  
*Static address.*
- uint16\_t `vendorID`  
*Device vendor ID(manufacture ID).*
- uint32\_t `partNumber`  
*Device part number info.*
- uint8\_t `dcr`  
*Device characteristics register information.*
- uint8\_t `bcr`  
*Bus characteristics register information.*
- uint8\_t `hdrMode`  
*Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.*
- bool `nakAllRequest`  
*Whether to reply NAK to all requests except broadcast CCC.*
- bool `ignoreS0S1Error`  
*Whether to ignore S0/S1 error in SDR mode.*
- bool `offline`  
*Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.*
- bool `matchSlaveStartStop`  
*Whether to assert start/stop status only the time slave is addressed.*
- uint32\_t `maxWriteLength`  
*Maximum write length.*
- uint32\_t `maxReadLength`  
*Maximum read length.*

## Field Documentation

- (1) `bool i3c_slave_config_t::enableSlave`
- (2) `bool i3c_slave_config_t::isHotJoin`
- (3) `uint8_t i3c_slave_config_t::staticAddr`
- (4) `uint16_t i3c_slave_config_t::vendorID`
- (5) `uint8_t i3c_slave_config_t::dcr`
- (6) `uint8_t i3c_slave_config_t::bcr`
- (7) `uint8_t i3c_slave_config_t::hdrMode`
- (8) `bool i3c_slave_config_t::nakAllRequest`
- (9) `bool i3c_slave_config_t::ignoreS0S1Error`
- (10) `bool i3c_slave_config_t::offline`
- (11) `bool i3c_slave_config_t::matchSlaveStartStop`
- (12) `uint32_t i3c_slave_config_t::maxWriteLength`
- (13) `uint32_t i3c_slave_config_t::maxReadLength`

### 19.6.2.2 struct i3c\_slave\_transfer\_t

#### Data Fields

- `uint32_t event`  
*Reason the callback is being invoked.*
- `uint8_t * txData`  
*Transfer buffer.*
- `size_t txDataSize`  
*Transfer size.*
- `uint8_t * rxData`  
*Transfer buffer.*
- `size_t rxDataSize`  
*Transfer size.*
- `status_t completionStatus`  
*Success or error code describing how the transfer completed.*
- `size_t transferredCount`  
*Number of bytes actually transferred since start or last repeated start.*

## Field Documentation

- (1) `uint32_t i3c_slave_transfer_t::event`
- (2) `status_t i3c_slave_transfer_t::completionStatus`  
Only applies for [kI3C\\_SlaveCompletionEvent](#).
- (3) `size_t i3c_slave_transfer_t::transferredCount`

### 19.6.2.3 struct \_i3c\_slave\_handle

Note

The contents of this structure are private and subject to change.

## Data Fields

- `i3c_slave_transfer_t transfer`  
*I3C slave transfer copy.*
- `bool isBusy`  
*Whether transfer is busy.*
- `bool wasTransmit`  
*Whether the last transfer was a transmit.*
- `uint32_t eventMask`  
*Mask of enabled events.*
- `uint32_t transferredCount`  
*Count of bytes transferred.*
- `i3c_slave_transfer_callback_t callback`  
*Callback function called at transfer event.*
- `void *userData`  
*Callback parameter passed to callback.*
- `uint8_t txFifoSize`  
*Tx Fifo size.*

## Field Documentation

- (1) `i3c_slave_transfer_t i3c_slave_handle_t::transfer`
- (2) `bool i3c_slave_handle_t::isBusy`
- (3) `bool i3c_slave_handle_t::wasTransmit`
- (4) `uint32_t i3c_slave_handle_t::eventMask`
- (5) `uint32_t i3c_slave_handle_t::transferredCount`
- (6) `i3c_slave_transfer_callback_t i3c_slave_handle_t::callback`
- (7) `void* i3c_slave_handle_t::userData`

## 19.6.3 Typedef Documentation

### 19.6.3.1 `typedef void(* i3c_slave_transfer_callback_t)(I3C_Type *base, i3c_slave_transfer_t *transfer, void *userData)`

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C\_SlaveSetCallback() function after you have created a handle.

## Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for the I3C instance on which the event occurred.                       |
| <i>transfer</i> | Pointer to transfer descriptor containing values passed to and/or from the callback. |
| <i>userData</i> | Arbitrary pointer-sized value passed from the application.                           |

**19.6.3.2 `typedef void(* i3c_slave_isr_t)(I3C_Type *base, void *handle)`****19.6.4 Enumeration Type Documentation****19.6.4.1 `enum _i3c_slave_flags`**

The following status register flags can be cleared:

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`

Only below flags can be enabled as interrupts.

- `kI3C_SlaveBusStartFlag`
- `kI3C_SlaveMatchedFlag`
- `kI3C_SlaveBusStopFlag`
- `kI3C_SlaveRxReadyFlag`
- `kI3C_SlaveTxReadyFlag`
- `kI3C_SlaveDynamicAddrChangedFlag`
- `kI3C_SlaveReceivedCCCFlag`
- `kI3C_SlaveErrorFlag`
- `kI3C_SlaveHDRCommandMatchFlag`
- `kI3C_SlaveCCCHandledFlag`
- `kI3C_SlaveEventSentFlag`

## Note

These enums are meant to be OR'd together to form a bit mask.

## Enumerator

**`kI3C_SlaveNotStopFlag`** Slave status not stop flag.

**`kI3C_SlaveMessageFlag`** Slave status message, indicating slave is listening to the bus traffic or responding.

**`kI3C_SlaveRequiredReadFlag`** Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

**`kI3C_SlaveRequiredWriteFlag`** Slave status request write, master is doing SDR write to slave, except slave in ENTDAA mode.

- kI3C\_SlaveBusDAAFlag* I3C bus is in ENTDAA mode.
- kI3C\_SlaveBusHDRModeFlag* I3C bus is in HDR mode.
- kI3C\_SlaveBusStartFlag* Start/Re-start event is seen since the bus was last cleared.
- kI3C\_SlaveMatchedFlag* Slave address(dynamic/static) matched since last cleared.
- kI3C\_SlaveBusStopFlag* Stop event is seen since the bus was last cleared.
- kI3C\_SlaveRxReadyFlag* Rx data ready in rx buffer flag.
- kI3C\_SlaveTxReadyFlag* Tx buffer ready for Tx data flag.
- kI3C\_SlaveDynamicAddrChangedFlag* Slave dynamic address has been assigned, re-assigned, or lost.
- kI3C\_SlaveReceivedCCCFlag* Slave received Common command code.
- kI3C\_SlaveErrorFlag* Error occurred flag.
- kI3C\_SlaveHDRCommandMatchFlag* High data rate command match.
- kI3C\_SlaveCCCHandledFlag* Slave received Common command code is handled by I3C module.
- kI3C\_SlaveEventSentFlag* Slave IBI/P2P/MR/HJ event has been sent.
- kI3C\_SlaveIbiDisableFlag* Slave in band interrupt is disabled.
- kI3C\_SlaveMasterRequestDisabledFlag* Slave master request is disabled.
- kI3C\_SlaveHotJoinDisabledFlag* Slave Hot-Join is disabled.
- kI3C\_SlaveClearFlags* All flags which are cleared by the driver upon starting a transfer.

#### 19.6.4.2 enum \_i3c\_slave\_error\_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

- kI3C\_SlaveErrorOverrunFlag* Slave internal from-bus buffer/FIFO overrun.
- kI3C\_SlaveErrorUnderrunFlag* Slave internal to-bus buffer/FIFO underrun.
- kI3C\_SlaveErrorUnderrunNakFlag* Slave internal from-bus buffer/FIFO underrun and NACK error.
- kI3C\_SlaveErrorTermFlag* Terminate error from master.
- kI3C\_SlaveErrorInvalidStartFlag* Slave invalid start flag.
- kI3C\_SlaveErrorSdrParityFlag* SDR parity error.
- kI3C\_SlaveErrorHdrParityFlag* HDR parity error.
- kI3C\_SlaveErrorHdrCRCFlag* HDR-DDR CRC error.
- kI3C\_SlaveErrorS0S1Flag* S0 or S1 error.
- kI3C\_SlaveErrorOverreadFlag* Over-read error.
- kI3C\_SlaveErrorOverwriteFlag* Over-write error.

#### 19.6.4.3 enum i3c\_slave\_event\_t

Enumerator

- kI3C\_SlaveEventNormal* Normal mode.

- kI3C\_SlaveEventIBI*** In band interrupt event.
- kI3C\_SlaveEventMasterReq*** Master request event.
- kI3C\_SlaveEventHotJoinReq*** Hot-join event.

#### 19.6.4.4 enum i3c\_slave\_activity\_state\_t

Enumerator

- kI3C\_SlaveNoLatency*** Normal bus operation.
- kI3C\_SlaveLatency1Ms*** 1ms of latency.
- kI3C\_SlaveLatency100Ms*** 100ms of latency.
- kI3C\_SlaveLatency10S*** 10s latency.

#### 19.6.4.5 enum i3c\_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 [I3C\\_SlaveTransferNonBlocking\(\)](#) in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

- kI3C\_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.
- kI3C\_SlaveTransmitEvent*** Callback is requested to provide data to transmit (slave-transmitter role).
- kI3C\_SlaveReceiveEvent*** Callback is requested to provide a buffer in which to place received data (slave-receiver role).
- kI3C\_SlaveRequiredTransmitEvent*** Callback is requested to provide a buffer in which to place received data (slave-receiver role).
- kI3C\_SlaveStartEvent*** A start/repeated start was detected.
- kI3C\_SlaveHDRCommandMatchEvent*** Slave Match HDR Command.
- kI3C\_SlaveCompletionEvent*** A stop was detected, completing the transfer.
- kI3C\_SlaveRequestSentEvent*** Slave request event sent.
- kI3C\_SlaveReceivedCCCEvent*** Slave received CCC event, need to handle by application.
- kI3C\_SlaveAllEvents*** Bit mask of all available events.

#### 19.6.5 Function Documentation

##### 19.6.5.1 void I3C\_SlaveGetDefaultConfig ( i3c\_slave\_config\_t \* *slaveConfig* )

This function provides the following default configuration for the I3C slave peripheral:

```
* slaveConfig->enableslave = true;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with [I3C\\_SlaveInit\(\)](#).

Parameters

|     |                    |                                                                                                         |
|-----|--------------------|---------------------------------------------------------------------------------------------------------|
| out | <i>slaveConfig</i> | User provided configuration structure for default values. Refer to <a href="#">i3c_slave_config_t</a> . |
|-----|--------------------|---------------------------------------------------------------------------------------------------------|

#### 19.6.5.2 void I3C\_SlaveInit ( I3C\_Type \* *base*, const i3c\_slave\_config\_t \* *slaveConfig*, uint32\_t *slowClock\_Hz* )

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

|                     |                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | The I3C peripheral base address.                                                                                                        |
| <i>slaveConfig</i>  | User provided peripheral configuration. Use <a href="#">I3C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>slowClock_Hz</i> | Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values.                                             |

#### 19.6.5.3 void I3C\_SlaveDeinit ( I3C\_Type \* *base* )

This function disables the I3C slave peripheral and gates the clock.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

#### 19.6.5.4 static void I3C\_SlaveEnable ( I3C\_Type \* *base*, bool *isEnable* ) [inline], [static]

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>base</i>     | The I3C peripheral base address. |
| <i>isEnable</i> | Enable or disable.               |

#### 19.6.5.5 static uint32\_t I3C\_SlaveGetStatusFlags ( I3C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_i3c\\_slave\\_flags](#)

#### 19.6.5.6 static void I3C\_SlaveClearStatusFlags ( I3C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- [kI3C\\_SlaveBusStartFlag](#)
- [kI3C\\_SlaveMatchedFlag](#)
- [kI3C\\_SlaveBusStopFlag](#)

Attempts to clear other flags has no effect.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

|                   |                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i3c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I3C_SlaveGetStatusFlags()</a> . |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

See Also

[\\_i3c\\_slave\\_flags](#).

#### 19.6.5.7 static uint32\_t I3C\_SlaveGetErrorStatusFlags ( I3C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_i3c\\_slave\\_error\\_flags](#)

#### 19.6.5.8 static void I3C\_SlaveClearErrorStatusFlags ( I3C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

Parameters

|                   |                                                                                                                                                                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I3C peripheral base address.                                                                                                                                                                                                                 |
| <i>statusMask</i> | A bitmask of error status flags that are to be cleared. The mask is composed of <a href="#">_i3c_slave_error_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I3C_SlaveGetErrorStatusFlags()</a> . |

See Also

[\\_i3c\\_slave\\_error\\_flags](#).

#### 19.6.5.9 i3c\_slave\_activity\_state\_t I3C\_SlaveGetActivityState ( I3C\_Type \* *base* )

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

I3C slave activity state, refer [i3c\\_slave\\_activity\\_state\\_t](#).

#### 19.6.5.10 static void I3C\_SlaveEnableInterrupts ( I3C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Only below flags can be enabled as interrupts.

- [kI3C\\_SlaveBusStartFlag](#)
- [kI3C\\_SlaveMatchedFlag](#)
- [kI3C\\_SlaveBusStopFlag](#)
- [kI3C\\_SlaveRxReadyFlag](#)
- [kI3C\\_SlaveTxReadyFlag](#)
- [kI3C\\_SlaveDynamicAddrChangedFlag](#)
- [kI3C\\_SlaveReceivedCCCFlag](#)
- [kI3C\\_SlaveErrorFlag](#)
- [kI3C\\_SlaveHDRCommandMatchFlag](#)
- [kI3C\\_SlaveCCCHandledFlag](#)
- [kI3C\\_SlaveEventSentFlag](#)

Parameters

|                      |                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I3C peripheral base address.                                                                                                                   |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_i3c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 19.6.5.11 static void I3C\_SlaveDisableInterrupts ( I3C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Only below flags can be disabled as interrupts.

- [kI3C\\_SlaveBusStartFlag](#)
- [kI3C\\_SlaveMatchedFlag](#)
- [kI3C\\_SlaveBusStopFlag](#)
- [kI3C\\_SlaveRxReadyFlag](#)
- [kI3C\\_SlaveTxReadyFlag](#)
- [kI3C\\_SlaveDynamicAddrChangedFlag](#)
- [kI3C\\_SlaveReceivedCCCFlag](#)

- kI3C\_SlaveErrorFlag
- kI3C\_SlaveHDRCommandMatchFlag
- kI3C\_SlaveCCCHandledFlag
- kI3C\_SlaveEventSentFlag

Parameters

|                      |                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I3C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_i3c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 19.6.5.12 static uint32\_t I3C\_SlaveGetEnabledInterrupts ( I3C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

A bitmask composed of [\\_i3c\\_slave\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

#### 19.6.5.13 static uint32\_t I3C\_SlaveGetPendingInterrupts ( I3C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

Returns

A bitmask composed of [\\_i3c\\_slave\\_flags](#) enumerators OR'd together to indicate the set of pending interrupts.

#### 19.6.5.14 static void I3C\_SlaveEnableDMA ( I3C\_Type \* *base*, bool *enableTx*, bool *enableRx*, uint32\_t *width* ) [inline], [static]

Parameters

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>base</i>     | The I3C peripheral base address.                                               |
| <i>enableTx</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |
| <i>enableRx</i> | Enable flag for receive DMA request. Pass true for enable, false for disable.  |
| <i>width</i>    | DMA read/write unit in bytes.                                                  |

**19.6.5.15 static uint32\_t I3C\_SlaveGetTxFifoAddress ( I3C\_Type \* *base*, uint32\_t *width* )  
[inline], [static]**

Parameters

|              |                                  |
|--------------|----------------------------------|
| <i>base</i>  | The I3C peripheral base address. |
| <i>width</i> | DMA read/write unit in bytes.    |

Returns

The I3C Slave Transmit Data Register address.

**19.6.5.16 static uint32\_t I3C\_SlaveGetRxFifoAddress ( I3C\_Type \* *base*, uint32\_t *width* )  
[inline], [static]**

Parameters

|              |                                  |
|--------------|----------------------------------|
| <i>base</i>  | The I3C peripheral base address. |
| <i>width</i> | DMA read/write unit in bytes.    |

Returns

The I3C Slave Receive Data Register address.

**19.6.5.17 static void I3C\_SlaveSetWatermarks ( I3C\_Type \* *base*, i3c\_tx\_trigger\_level\_t  
txLvl, i3c\_rx\_trigger\_level\_t rxLvl, bool flushTx, bool flushRx ) [inline],  
[static]**

Parameters

|                |                                                                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I3C peripheral base address.                                                                                                                              |
| <i>txLvl</i>   | Transmit FIFO watermark level. The <a href="#">kI3C_SlaveTxReadyFlag</a> flag is set whenever the number of words in the transmit FIFO reaches <i>txLvl</i> . |
| <i>rxLvl</i>   | Receive FIFO watermark level. The <a href="#">kI3C_SlaveRxReadyFlag</a> flag is set whenever the number of words in the receive FIFO reaches <i>rxLvl</i> .   |
| <i>flushTx</i> | true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.                                                                                        |
| <i>flushRx</i> | true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.                                                                                        |

**19.6.5.18 static void I3C\_SlaveGetFifoCounts ( I3C\_Type \* *base*, size\_t \* *rxCount*, size\_t \* *txCount* ) [inline], [static]**

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | The I3C peripheral base address.                                                                                             |
| out | <i>txCount</i> | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.  |

**19.6.5.19 void I3C\_SlaveRequestEvent ( I3C\_Type \* *base*, i3c\_slave\_event\_t *event* )**

Parameters

|              |                                                           |
|--------------|-----------------------------------------------------------|
| <i>base</i>  | The I3C peripheral base address.                          |
| <i>event</i> | I3C slave event of type <a href="#">i3c_slave_event_t</a> |

**19.6.5.20 status\_t I3C\_SlaveSend ( I3C\_Type \* *base*, const void \* *txBuff*, size\_t *txSize* )**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

Returns

Error or success status returned by API.

### 19.6.5.21 **status\_t I3C\_SlaveReceive ( I3C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize* )**

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I3C peripheral base address.                   |
| <i>rxBuff</i> | The pointer to the data to be transferred.         |
| <i>rxSize</i> | The length in bytes of the data to be transferred. |

Returns

Error or success status returned by API.

### 19.6.5.22 **void I3C\_SlaveTransferCreateHandle ( I3C\_Type \* *base*, i3c\_slave\_handle\_t \* *handle*, i3c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I3C\\_SlaveTransferAbort\(\)](#) API shall be called.

Note

The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

|            |                 |                                                              |
|------------|-----------------|--------------------------------------------------------------|
|            | <i>base</i>     | The I3C peripheral base address.                             |
| <i>out</i> | <i>handle</i>   | Pointer to the I3C slave driver handle.                      |
|            | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|            | <i>userData</i> | User provided pointer to the application callback data.      |

### 19.6.5.23 status\_t I3C\_SlaveTransferNonBlocking ( *I3C\_Type \* base*, *i3c\_slave\_handle\_t \* handle*, *uint32\_t eventMask* )

Call this API after calling `I2C_SlaveInit()` and `I3C_SlaveTransferCreateHandle()` to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to `I3C_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 `i3c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI3C_SlaveTransmitEvent` and `kI3C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI3C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

|                  |                                                                                                                                                                                                                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I3C peripheral base address.                                                                                                                                                                                                                                                             |
| <i>handle</i>    | Pointer to struct: <code>_i3c_slave_handle</code> structure which stores the transfer state.                                                                                                                                                                                                 |
| <i>eventMask</i> | Bit mask formed by OR'ing together <code>i3c_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>kI3C_SlaveAllEvents</code> to enable all events. |

Return values

|                               |                                                           |
|-------------------------------|-----------------------------------------------------------|
| <code>kStatus_Success</code>  | Slave transfers were successfully started.                |
| <code>kStatus_I3C_Busy</code> | Slave transfers have already been started on this handle. |

### 19.6.5.24 status\_t I3C\_SlaveTransferGetCount ( *I3C\_Type \* base*, *i3c\_slave\_handle\_t \* handle*, *size\_t \* count* )

Parameters

|            |               |                                                                                                       |
|------------|---------------|-------------------------------------------------------------------------------------------------------|
|            | <i>base</i>   | The I3C peripheral base address.                                                                      |
|            | <i>handle</i> | Pointer to <code>i2c_slave_handle_t</code> structure.                                                 |
| <i>out</i> | <i>count</i>  | Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required. |

Return values

|                                      |  |
|--------------------------------------|--|
| <i>kStatus_Success</i>               |  |
| <i>kStatus_NoTransferIn-Progress</i> |  |

### 19.6.5.25 void I3C\_SlaveTransferAbort ( **I3C\_Type** \* *base*, **i3c\_slave\_handle\_t** \* *handle* )

Note

This API could be called at any time to stop slave for handling the bus events.

Parameters

|               |                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I3C peripheral base address.                                                                       |
| <i>handle</i> | Pointer to struct: <a href="#"><i>_i3c_slave_handle</i></a> structure which stores the transfer state. |

Return values

|                         |  |
|-------------------------|--|
| <i>kStatus_Success</i>  |  |
| <i>kStatus_I3C_Idle</i> |  |

### 19.6.5.26 void I3C\_SlaveTransferHandleIRQ ( **I3C\_Type** \* *base*, **void** \* *intHandle* )

Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

|                  |                                                                                                        |
|------------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I3C peripheral base address.                                                                       |
| <i>intHandle</i> | Pointer to struct: <a href="#"><i>_i3c_slave_handle</i></a> structure which stores the transfer state. |

### 19.6.5.27 void I3C\_SlaveRequestIBIWithData ( **I3C\_Type** \* *base*, **uint8\_t** \* *data*, **size\_t** *dataSize* )

Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>base</i>     | The I3C peripheral base address.               |
| <i>data</i>     | Pointer to IBI data to be sent in the request. |
| <i>dataSize</i> | IBI data size.                                 |

**19.6.5.28 void I3C\_SlaveRequestIBIWithSingleData ( I3C\_Type \* *base*, uint8\_t *data*, size\_t *dataSize* )**

**Deprecated** Do not use this function. It has been superseded by [I3C\\_SlaveRequestIBIWithData](#).

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | The I3C peripheral base address.    |
| <i>data</i>     | IBI data to be sent in the request. |
| <i>dataSize</i> | IBI data size.                      |

## 19.7 I3C Common Driver

### 19.7.1 Overview

#### Data Structures

- struct [i3c\\_config\\_t](#)

*Structure with settings to initialize the I3C module, could both initialize master and slave functionality.*  
[More...](#)

#### Functions

- `uint32_t I3CGetInstance (I3C_Type *base)`  
*Get which instance current I3C is used.*
- `void I3CGetDefaultConfig (i3c_config_t *config)`  
*Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.*
- `void I3CInit (I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)`  
*Initializes the I3C peripheral.*

### 19.7.2 Data Structure Documentation

#### 19.7.2.1 struct i3c\_config\_t

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the [I3CGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

#### Data Fields

- `i3c_master_enable_t enableMaster`  
*Enable master mode.*
- `bool disableTimeout`  
*Whether to disable timeout to prevent the ERRWARN.*
- `i3c_master_hkeep_t hKeep`  
*High keeper mode setting.*
- `bool enableOpenDrainStop`  
*Whether to emit open-drain speed STOP.*
- `bool enableOpenDrainHigh`  
*Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.*
- `i3c_baudrate_hz_t baudRate_Hz`  
*Desired baud rate settings.*
- `uint8_t masterDynamicAddress`  
*Main master dynamic address configuration.*

- **uint32\_t slowClock\_Hz**  
*Slow clock frequency for time control.*
- **uint32\_t maxWriteLength**  
*Maximum write length.*
- **uint32\_t maxReadLength**  
*Maximum read length.*
- **bool enableSlave**  
*Whether to enable slave.*
- **uint8\_t staticAddr**  
*Static address.*
- **uint16\_t vendorID**  
*Device vendor ID(manufacture ID).*
- **uint32\_t partNumber**  
*Device part number info.*
- **uint8\_t dcr**  
*Device characteristics register information.*
- **uint8\_t bcr**  
*Bus characteristics register information.*
- **uint8\_t hdrMode**  
*Support hdr mode, could be OR logic in enumeration:`i3c_hdr_mode_t`.*
- **bool nakAllRequest**  
*Whether to reply NAK to all requests except broadcast CCC.*
- **bool ignoreS0S1Error**  
*Whether to ignore S0/S1 error in SDR mode.*
- **bool offline**  
*Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.*
- **bool matchSlaveStartStop**  
*Whether to assert start/stop status only the time slave is addressed.*

## Field Documentation

- (1) `i3c_master_enable_t i3c_config_t::enableMaster`
- (2) `bool i3c_config_t::disableTimeout`
- (3) `i3c_master_hkeep_t i3c_config_t::hKeep`
- (4) `bool i3c_config_t::enableOpenDrainStop`
- (5) `bool i3c_config_t::enableOpenDrainHigh`
- (6) `i3c_baudrate_hz_t i3c_config_t::baudRate_Hz`
- (7) `uint8_t i3c_config_t::masterDynamicAddress`
- (8) `uint32_t i3c_config_t::slowClock_Hz`
- (9) `uint32_t i3c_config_t::maxWriteLength`
- (10) `uint32_t i3c_config_t::maxReadLength`
- (11) `bool i3c_config_t::enableSlave`
- (12) `uint8_t i3c_config_t::staticAddr`
- (13) `uint16_t i3c_config_t::vendorID`
- (14) `uint8_t i3c_config_t::dcr`
- (15) `uint8_t i3c_config_t::bcr`
- (16) `uint8_t i3c_config_t::hdrMode`
- (17) `bool i3c_config_t::nakAllRequest`
- (18) `bool i3c_config_t::ignoreS0S1Error`
- (19) `bool i3c_config_t::offline`
- (20) `bool i3c_config_t::matchSlaveStartStop`

## 19.7.3 Function Documentation

### 19.7.3.1 `uint32_t I3C_GetInstance ( I3C_Type * base )`

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I3C peripheral base address. |
|-------------|----------------------------------|

**19.7.3.2 void I3C\_GetDefaultConfig ( i3c\_config\_t \* *config* )**

This function provides the following default configuration for I3C:

```
* config->enableMaster = kI3C_MasterCapable;
* config->disableTimeout = false;
* config->hKeep = kI3C_MasterHighKeeperNone;
* config->enableOpenDrainStop = true;
* config->enableOpenDrainHigh = true;
* config->baudRate_Hz.i2cBaud = 400000U;
* config->baudRate_Hz.i3cPushPullBaud = 12500000U;
* config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
* config->masterDynamicAddress = 0x0AU;
* config->slowClock_Hz = 1000000U;
* config->enableSlave = true;
* config->vendorID = 0x11BU;
* config->enableRandomPart = false;
* config->partNumber = 0;
* config->dcr = 0;
* config->bcr = 0;
* config->hdrMode = (uint8_t)kI3C_HDRModeDDR;
* config->nakAllRequest = false;
* config->ignoreS0S1Error = false;
* config->offline = false;
* config->matchSlaveStartStop = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with [I3C\\_Init\(\)](#).

## Parameters

|            |               |                                                                                                   |
|------------|---------------|---------------------------------------------------------------------------------------------------|
| <i>out</i> | <i>config</i> | User provided configuration structure for default values. Refer to <a href="#">i3c_config_t</a> . |
|------------|---------------|---------------------------------------------------------------------------------------------------|

**19.7.3.3 void I3C\_Init ( I3C\_Type \* *base*, const i3c\_config\_t \* *config*, uint32\_t *sourceClock\_Hz* )**

This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

## Parameters

|                        |                                                                                                                                    |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>            | The I3C peripheral base address.                                                                                                   |
| <i>config</i>          | User provided peripheral configuration. Use <a href="#">I3C_GetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_-Hz</i> | Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.      |

# Chapter 20

## KPP: KeyPad Port Driver

### 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the KeyPad Port block of MCUXpresso SDK devices.

#### KPP: KeyPad Port Driver

##### KPP Initialization Operation

The KPP Initialize is to initialize for common configure: gate the KPP clock, configure columns, and rows features. The KPP Deinitialize is to ungate the clock.

##### KPP Basic Operation

The KPP provide the function to enable/disable interrupts. The KPP provide key press scanning function KPP\_keyPressScanning. This API should be called by the Interrupt handler in application. KPP still provides functions to get and clear status flags.

### 20.2 Typical use case

#### Data Structures

- struct [kpp\\_config\\_t](#)  
*Lists of KPP status. [More...](#)*

#### Enumerations

- enum [kpp\\_interrupt\\_enable\\_t](#) {  
  kKPP\_keyDepressInterrupt = KPP\_KPSR\_KDIE\_MASK,  
  kKPP\_keyReleaseInterrupt = KPP\_KPSR\_KRIE\_MASK }  
*List of interrupts supported by the peripheral.*
- enum [kpp\\_sync\\_operation\\_t](#) {  
  kKPP\_ClearKeyDepressSyncChain = KPP\_KPSR\_KDSC\_MASK,  
  kKPP\_SetKeyReleasesSyncChain = KPP\_KPSR\_KRSS\_MASK }  
*Lists of KPP synchronize chain operation.*

#### Driver version

- #define [FSL\\_KPP\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*KPP driver version 2.0.0.*

## Initialization and De-initialization

- void **KPP\_Init** (KPP\_Type \*base, **kpp\_config\_t** \*configure)  
*KPP initialize.*
- void **KPP\_Deinit** (KPP\_Type \*base)  
*Deinitializes the KPP module and gates the clock.*

## KPP Basic Operation

- static void **KPP\_EnableInterrupts** (KPP\_Type \*base, uint16\_t mask)  
*Enable the interrupt.*
- static void **KPP\_DisableInterrupts** (KPP\_Type \*base, uint16\_t mask)  
*Disable the interrupt.*
- static uint16\_t **KPP\_GetStatusFlag** (KPP\_Type \*base)  
*Gets the KPP interrupt event status.*
- static void **KPP\_ClearStatusFlag** (KPP\_Type \*base, uint16\_t mask)  
*Clears KPP status flag.*
- static void **KPP\_SetSynchronizeChain** (KPP\_Type \*base, uint16\_t mask)  
*Set KPP synchronization chain.*
- void **KPP\_KeyPressScanning** (KPP\_Type \*base, uint8\_t \*data, uint32\_t clockSrc\_Hz)  
*Keypad press scanning.*

## 20.3 Data Structure Documentation

### 20.3.1 struct kpp\_config\_t

#### Data Fields

- uint8\_t **activeRow**  
*The row number: bit 7 ~ 0 represents the row 7 ~ 0.*
- uint8\_t **activeColumn**  
*The column number: bit 7 ~ 0 represents the column 7 ~ 0.*
- uint16\_t **interrupt**  
*KPP interrupt source.*

#### Field Documentation

- (1) **uint8\_t kpp\_config\_t::activeRow**
- (2) **uint8\_t kpp\_config\_t::activeColumn**
- (3) **uint16\_t kpp\_config\_t::interrupt**

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

## 20.4 Macro Definition Documentation

### 20.4.1 #define FSL\_KPP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 20.5 Enumeration Type Documentation

### 20.5.1 enum kpp\_interrupt\_enable\_t

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

Enumerator

- kKPP\_keyDepressInterrupt* Keypad depress interrupt source.
- kKPP\_keyReleaseInterrupt* Keypad release interrupt source.

### 20.5.2 enum kpp\_sync\_operation\_t

Enumerator

- kKPP\_ClearKeyDepressSyncChain* Keypad depress interrupt status.
- kKPP\_SetKeyReleasesSyncChain* Keypad release interrupt status.

## 20.6 Function Documentation

### 20.6.1 void KPP\_Init ( **KPP\_Type** \* *base*, **kpp\_config\_t** \* *configure* )

This function ungates the KPP clock and initializes KPP. This function must be called before calling any other KPP driver functions.

Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | KPP peripheral base address.             |
| <i>configure</i> | The KPP configuration structure pointer. |

### 20.6.2 void KPP\_Deinit ( **KPP\_Type** \* *base* )

This function gates the KPP clock. As a result, the KPP module doesn't work after calling this function.

Parameters

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

### 20.6.3 static void KPP\_EnableInterrupts ( **KPP\_Type** \* *base*, **uint16\_t** *mask* ) [**inline**], [**static**]

Parameters

|             |                                                                                              |
|-------------|----------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                                 |
| <i>mask</i> | KPP interrupts to enable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |

#### 20.6.4 static void KPP\_DisableInterrupts ( **KPP\_Type** \* *base*, **uint16\_t** *mask* ) [**inline**], [**static**]

Parameters

|             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                                  |
| <i>mask</i> | KPP interrupts to disable. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |

#### 20.6.5 static **uint16\_t** KPP\_GetStatusFlag ( **KPP\_Type** \* *base* ) [**inline**], [**static**]

Parameters

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

Returns

The status of the KPP. Application can use the enum type in the "kpp\_interrupt\_enable\_t" to get the right status of the related event.

#### 20.6.6 static void KPP\_ClearStatusFlag ( **KPP\_Type** \* *base*, **uint16\_t** *mask* ) [**inline**], [**static**]

Parameters

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

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| <i>mask</i> | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_interrupt_enable_t. |
|-------------|--------------------------------------------------------------------------------------------|

### 20.6.7 static void KPP\_SetSynchronizeChain ( KPP\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                          |
|-------------|------------------------------------------------------------------------------------------|
| <i>base</i> | KPP peripheral base address.                                                             |
| <i>mask</i> | KPP mask to be cleared. This is a logical OR of the enumeration :: kpp_sync_operation_t. |

### 20.6.8 void KPP\_keyPressScanning ( KPP\_Type \* *base*, uint8\_t \* *data*, uint32\_t *clockSrc\_Hz* )

This function will scanning all columns and rows. so all scanning data will be stored in the data pointer.

Parameters

|                    |                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | KPP peripheral base address.                                                                                                                                                                                                                                                                                                |
| <i>data</i>        | KPP key press scanning data. The data buffer should be prepared with length at least equal to KPP_KEYPAD_COLUMNNUM_MAX * KPP_KEYPAD_ROWNUM_MAX. the data pointer is recommended to be a array like uint8_t data[KPP_KEYPAD_COLUMNNUM_MAX]. for example the data[2] = 4, that means in column 1 row 2 has a key press event. |
| <i>clockSrc_Hz</i> | Source clock.                                                                                                                                                                                                                                                                                                               |

# Chapter 21

## LPADC: 12-bit SAR Analog-to-Digital Converter Driver

### 21.1 Overview

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

### 21.2 Typical use case

#### 21.2.1 Polling Configuration

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

#### 21.2.2 Interrupt Configuration

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

### Files

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

### Data Structures

- struct [lpadc\\_config\\_t](#)  
*LPADC global configuration. [More...](#)*
- struct [lpadc\\_conv\\_command\\_config\\_t](#)  
*Define structure to keep the configuration for conversion command. [More...](#)*
- struct [lpadc\\_conv\\_trigger\\_config\\_t](#)  
*Define structure to keep the configuration for conversion trigger. [More...](#)*
- struct [lpadc\\_conv\\_result\\_t](#)  
*Define the structure to keep the conversion result. [More...](#)*
- struct [lpadc\\_calibration\\_value\\_t](#)  
*A structure of calibration value. [More...](#)*

### Macros

- #define [LPADC\\_GET\\_ACTIVE\\_COMMAND\\_STATUS](#)(statusVal) ((statusVal & ADC\_STAT\_CMMDACT\_MASK) >> ADC\_STAT\_CMMDACT\_SHIFT)  
*Define the MACRO function to get command status from status value.*
- #define [LPADC\\_GET\\_ACTIVE\\_TRIGGER\\_STATUS](#)(statusVal) ((statusVal & ADC\_STAT\_TRGACT\_MASK) >> ADC\_STAT\_TRGACT\_SHIFT)  
*Define the MACRO function to get trigger status from status value.*

## Enumerations

- enum \_lpadc\_status\_flags {
   
kLPADC\_ResultFIFO0OverflowFlag = ADC\_STAT\_FOF0\_MASK,  
 kLPADC\_ResultFIFO0ReadyFlag = ADC\_STAT\_RDY0\_MASK,  
 kLPADC\_ResultFIFO1OverflowFlag = ADC\_STAT\_FOF1\_MASK,  
 kLPADC\_ResultFIFO1ReadyFlag = ADC\_STAT\_RDY1\_MASK,  
 kLPADC\_TriggerExceptionFlag = ADC\_STAT\_TEXC\_INT\_MASK,  
 kLPADC\_TriggerCompletionFlag = ADC\_STAT\_TCOMP\_INT\_MASK,  
 kLPADC\_CalibrationReadyFlag = ADC\_STAT\_CAL\_RDY\_MASK,  
 kLPADC\_ActiveFlag = ADC\_STAT\_ADC\_ACTIVE\_MASK,  
 kLPADC\_ResultFIFOOverflowFlag = kLPADC\_ResultFIFO0OverflowFlag,  
 kLPADC\_ResultFIFOReadyFlag = kLPADC\_ResultFIFO0ReadyFlag }
   
*Define hardware flags of the module.*
- enum \_lpadc\_interrupt\_enable {
   
kLPADC\_ResultFIFO0OverflowInterruptEnable = ADC\_IE\_FOFIE0\_MASK,  
 kLPADC\_FIFO0WatermarkInterruptEnable = ADC\_IE\_FWMIE0\_MASK,  
 kLPADC\_ResultFIFOOverflowInterruptEnable = kLPADC\_ResultFIFO0OverflowInterruptEnable,  
 kLPADC\_FIFOWatermarkInterruptEnable = kLPADC\_FIFO0WatermarkInterruptEnable,  
 kLPADC\_ResultFIFO1OverflowInterruptEnable = ADC\_IE\_FOFIE1\_MASK,  
 kLPADC\_FIFO1WatermarkInterruptEnable = ADC\_IE\_FWMIE1\_MASK,  
 kLPADC\_TriggerExceptionInterruptEnable = ADC\_IE\_TEXC\_IE\_MASK,  
 kLPADC\_Trigger0CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 0UL),  
 kLPADC\_Trigger1CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 1UL),  
 kLPADC\_Trigger2CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 2UL),  
 kLPADC\_Trigger3CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 3UL),  
 kLPADC\_Trigger4CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 4UL),  
 kLPADC\_Trigger5CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 5UL),  
 kLPADC\_Trigger6CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 6UL),  
 kLPADC\_Trigger7CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 7UL),  
 kLPADC\_Trigger8CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 8UL),  
 kLPADC\_Trigger9CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 9UL),  
 kLPADC\_Trigger10CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 10UL),  
 kLPADC\_Trigger11CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 11UL),  
 kLPADC\_Trigger12CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 12UL),  
 kLPADC\_Trigger13CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 13UL),  
 kLPADC\_Trigger14CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 14UL),  
 kLPADC\_Trigger15CompletionInterruptEnable = ADC\_IE\_TCOMP\_IE(1UL << 15UL) }
   
*Define interrupt switchers of the module.*
- enum \_lpadc\_trigger\_status\_flags {

```

kLPADC_Trigger0InterruptedFlag = 1UL << 0UL,
kLPADC_Trigger1InterruptedFlag = 1UL << 1UL,
kLPADC_Trigger2InterruptedFlag = 1UL << 2UL,
kLPADC_Trigger3InterruptedFlag = 1UL << 3UL,
kLPADC_Trigger4InterruptedFlag = 1UL << 4UL,
kLPADC_Trigger5InterruptedFlag = 1UL << 5UL,
kLPADC_Trigger6InterruptedFlag = 1UL << 6UL,
kLPADC_Trigger7InterruptedFlag = 1UL << 7UL,
kLPADC_Trigger8InterruptedFlag = 1UL << 8UL,
kLPADC_Trigger9InterruptedFlag = 1UL << 9UL,
kLPADC_Trigger10InterruptedFlag = 1UL << 10UL,
kLPADC_Trigger11InterruptedFlag = 1UL << 11UL,
kLPADC_Trigger12InterruptedFlag = 1UL << 12UL,
kLPADC_Trigger13InterruptedFlag = 1UL << 13UL,
kLPADC_Trigger14InterruptedFlag = 1UL << 14UL,
kLPADC_Trigger15InterruptedFlag = 1UL << 15UL,
kLPADC_Trigger0CompletedFlag = 1UL << 16UL,
kLPADC_Trigger1CompletedFlag = 1UL << 17UL,
kLPADC_Trigger2CompletedFlag = 1UL << 18UL,
kLPADC_Trigger3CompletedFlag = 1UL << 19UL,
kLPADC_Trigger4CompletedFlag = 1UL << 20UL,
kLPADC_Trigger5CompletedFlag = 1UL << 21UL,
kLPADC_Trigger6CompletedFlag = 1UL << 22UL,
kLPADC_Trigger7CompletedFlag = 1UL << 23UL,
kLPADC_Trigger8CompletedFlag = 1UL << 24UL,
kLPADC_Trigger9CompletedFlag = 1UL << 25UL,
kLPADC_Trigger10CompletedFlag = 1UL << 26UL,
kLPADC_Trigger11CompletedFlag = 1UL << 27UL,
kLPADC_Trigger12CompletedFlag = 1UL << 28UL,
kLPADC_Trigger13CompletedFlag = 1UL << 29UL,
kLPADC_Trigger14CompletedFlag = 1UL << 30UL,
kLPADC_Trigger15CompletedFlag = 1UL << 31UL }

```

*The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.*

- enum `lpadc_sample_scale_mode_t` {
   
  `kLPADC_SamplePartScale` = 0U,
   
  `kLPADC_SampleFullScale` = 1U }
- Define enumeration of sample scale mode.*
- enum `lpadc_sample_channel_mode_t` {
   
  `kLPADC_SampleChannelSingleEndSideA` = 0x0U,
   
  `kLPADC_SampleChannelSingleEndSideB` = 0x1U,
   
  `kLPADC_SampleChannelDiffBothSide` = 0x02U,
   
  `kLPADC_SampleChannelDualSingleEndBothSide` = 0x03U }
- Define enumeration of channel sample mode.*
- enum `lpadc_hardware_average_mode_t` {

```
kLPADC_HardwareAverageCount1 = 0U,
kLPADC_HardwareAverageCount2 = 1U,
kLPADC_HardwareAverageCount4 = 2U,
kLPADC_HardwareAverageCount8 = 3U,
kLPADC_HardwareAverageCount16 = 4U,
kLPADC_HardwareAverageCount32 = 5U,
kLPADC_HardwareAverageCount64 = 6U,
kLPADC_HardwareAverageCount128 = 7U,
kLPADC_HardwareAverageCount256 = 8U,
kLPADC_HardwareAverageCount512 = 9U,
kLPADC_HardwareAverageCount1024 = 10U }
```

*Define enumeration of hardware average selection.*

- enum `lpadc_sample_time_mode_t` {
 

```
kLPADC_SampleTimeADCK3 = 0U,
kLPADC_SampleTimeADCK5 = 1U,
kLPADC_SampleTimeADCK7 = 2U,
kLPADC_SampleTimeADCK11 = 3U,
kLPADC_SampleTimeADCK19 = 4U,
kLPADC_SampleTimeADCK35 = 5U,
kLPADC_SampleTimeADCK67 = 6U,
kLPADC_SampleTimeADCK131 = 7U }
```

*Define enumeration of sample time selection.*

- enum `lpadc_hardware_compare_mode_t` {
 

```
kLPADC_HardwareCompareDisabled = 0U,
kLPADC_HardwareCompareStoreOnTrue = 2U,
kLPADC_HardwareCompareRepeatUntilTrue = 3U }
```

*Define enumeration of hardware compare mode.*

- enum `lpadc_conversion_resolution_mode_t` {
 

```
kLPADC_ConversionResolutionStandard = 0U,
kLPADC_ConversionResolutionHigh = 1U }
```

*Define enumeration of conversion resolution mode.*

- enum `lpadc_conversion_average_mode_t` {
 

```
kLPADC_ConversionAverage1 = 0U,
kLPADC_ConversionAverage2 = 1U,
kLPADC_ConversionAverage4 = 2U,
kLPADC_ConversionAverage8 = 3U,
kLPADC_ConversionAverage16 = 4U,
kLPADC_ConversionAverage32 = 5U,
kLPADC_ConversionAverage64 = 6U,
kLPADC_ConversionAverage128 = 7U,
kLPADC_ConversionAverage256 = 8U,
kLPADC_ConversionAverage512 = 9U,
kLPADC_ConversionAverage1024 = 10U }
```

*Define enumeration of conversion averages mode.*

- enum `lpadc_reference_voltage_source_t` {

```

kLPADC_ReferenceVoltageAlt1 = 0U,
kLPADC_ReferenceVoltageAlt2 = 1U,
kLPADC_ReferenceVoltageAlt3 = 2U }

Define enumeration of reference voltage source.
• enum lpadc_power_level_mode_t {
 kLPADC_PowerLevelAlt1 = 0U,
 kLPADC_PowerLevelAlt2 = 1U,
 kLPADC_PowerLevelAlt3 = 2U,
 kLPADC_PowerLevelAlt4 = 3U }

Define enumeration of power configuration.
• enum lpadc_offset_calibration_mode_t {
 kLPADC_OffsetCalibration12bitMode = 0U,
 kLPADC_OffsetCalibration16bitMode = 1U }

Define enumeration of offset calibration mode.
• enum lpadc_trigger_priority_policy_t {
 kLPADC_ConvPreemptImmediatelyNotAutoResumed = 0x0U,
 kLPADC_ConvPreemptSoftlyNotAutoResumed = 0x1U,
 kLPADC_ConvPreemptImmediatelyAutoRestarted = 0x4U,
 kLPADC_ConvPreemptSoftlyAutoRestarted = 0x5U,
 kLPADC_ConvPreemptImmediatelyAutoResumed = 0xCU,
 kLPADC_ConvPreemptSoftlyAutoResumed = 0xDU,
 kLPADC_TriggerPriorityPreemptImmediately,
 kLPADC_TriggerPriorityPreemptSoftly,
 kLPADC_ConvPreemptSubsequentlyNotAutoResumed = 0x2U,
 kLPADC_ConvPreemptSubsequentlyAutoRestarted = 0x6U,
 kLPADC_ConvPreemptSubsequentlyAutoResumed = 0xEU,
 kLPADC_TriggerPriorityPreemptSubsequently,
 kLPADC_TriggerPriorityExceptionDisabled = 0x10U }

Define enumeration of trigger priority policy.

```

## Driver version

- #define FSL\_LPADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 8, 4))  
*LPADC driver version 2.8.4.*

## Initialization & de-initialization.

- void LPADC\_Init (ADC\_Type \*base, const lpadc\_config\_t \*config)  
*Initializes the LPADC module.*
- void LPADC\_GetDefaultConfig (lpadc\_config\_t \*config)  
*Gets an available pre-defined settings for initial configuration.*
- void LPADC\_Deinit (ADC\_Type \*base)  
*De-initializes the LPADC module.*
- static void LPADC\_Enable (ADC\_Type \*base, bool enable)  
*Switch on/off the LPADC module.*
- static void LPADC\_DoResetFIFO0 (ADC\_Type \*base)  
*Do reset the conversion FIFO0.*
- static void LPADC\_DoResetFIFO1 (ADC\_Type \*base)

*Do reset the conversion FIFO1.*

- static void [LPADC\\_DoResetConfig](#) (ADC\_Type \*base)  
*Do reset the module's configuration.*

## Status

- static uint32\_t [LPADC\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Get status flags.*
- static void [LPADC\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clear status flags.*
- static uint32\_t [LPADC\\_GetTriggerStatusFlags](#) (ADC\_Type \*base)  
*Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.*
- static void [LPADC\\_ClearTriggerStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clear trigger status flags.*

## Interrupts

- static void [LPADC\\_EnableInterrupts](#) (ADC\_Type \*base, uint32\_t mask)  
*Enable interrupts.*
- static void [LPADC\\_DisableInterrupts](#) (ADC\_Type \*base, uint32\_t mask)  
*Disable interrupts.*

## DMA Control

- static void [LPADC\\_EnableFIFO0WatermarkDMA](#) (ADC\_Type \*base, bool enable)  
*Switch on/off the DMA trigger for FIFO0 watermark event.*
- static void [LPADC\\_EnableFIFO1WatermarkDMA](#) (ADC\_Type \*base, bool enable)  
*Switch on/off the DMA trigger for FIFO1 watermark event.*

## Trigger and conversion with FIFO.

- static uint32\_t [LPADC\\_GetConvResultCount](#) (ADC\_Type \*base, uint8\_t index)  
*Get the count of result kept in conversion FIFO<sub>n</sub>.*
- bool [LPADC\\_GetConvResult](#) (ADC\_Type \*base, [lpadc\\_conv\\_result\\_t](#) \*result, uint8\_t index)  
*Get the result in conversion FIFO<sub>n</sub>.*
- void [LPADC\\_GetConvResultBlocking](#) (ADC\_Type \*base, [lpadc\\_conv\\_result\\_t](#) \*result, uint8\_t index)  
*Get the result in conversion FIFO<sub>n</sub> using blocking method.*
- void [LPADC\\_SetConvTriggerConfig](#) (ADC\_Type \*base, uint32\_t triggerId, const [lpadc\\_conv\\_trigger\\_config\\_t](#) \*config)  
*Configure the conversion trigger source.*
- void [LPADC\\_GetDefaultConvTriggerConfig](#) ([lpadc\\_conv\\_trigger\\_config\\_t](#) \*config)  
*Gets an available pre-defined settings for trigger's configuration.*
- static void [LPADC\\_DoSoftwareTrigger](#) (ADC\_Type \*base, uint32\_t triggerIdMask)  
*Do software trigger to conversion command.*
- void [LPADC\\_SetConvCommandConfig](#) (ADC\_Type \*base, uint32\_t commandId, const [lpadc\\_conv\\_command\\_config\\_t](#) \*config)  
*Configure conversion command.*
- void [LPADC\\_GetDefaultConvCommandConfig](#) ([lpadc\\_conv\\_command\\_config\\_t](#) \*config)

- Gets an available pre-defined settings for conversion command's configuration.
- static void **LPADC\_SetOffset12BitValue** (ADC\_Type \*base, uint32\_t valueA, uint32\_t valueB)
 

*Set proper offset value to trim 12 bit ADC conversion.*
- static void **LPADC\_SetOffset16BitValue** (ADC\_Type \*base, uint32\_t valueA, uint32\_t valueB)
 

*Set proper offset value to trim 16 bit ADC conversion.*
- static void **LPADC\_EnableOffsetCalibration** (ADC\_Type \*base, bool enable)
 

*Enable the offset calibration function.*
- static void **LPADC\_SetOffsetCalibrationMode** (ADC\_Type \*base, **lpadc\_offset\_calibration\_mode\_t** mode)
 

*Set offset calibration mode.*
- void **LPADC\_DoOffsetCalibration** (ADC\_Type \*base)
 

*Do offset calibration.*
- void **LPADC\_DoAutoCalibration** (ADC\_Type \*base)
 

*Do auto calibration.*
- void **LPADC\_PrepareAutoCalibration** (ADC\_Type \*base)
 

*Prepare auto calibration, LPADC\_FinishAutoCalibration has to be called before using the LPADC.*
- void **LPADC\_FinishAutoCalibration** (ADC\_Type \*base)
 

*Finish auto calibration start with LPADC\_PrepareAutoCalibration.*
- void **LPADC\_GetCalibrationValue** (ADC\_Type \*base, **lpadc\_calibration\_value\_t** \*ptrCalibrationValue)
 

*Get calibration value into the memory which is defined by invoker.*
- void **LPADC\_SetCalibrationValue** (ADC\_Type \*base, const **lpadc\_calibration\_value\_t** \*ptrCalibrationValue)
 

*Set calibration value into ADC calibration registers.*

## 21.3 Data Structure Documentation

### 21.3.1 struct **lpadc\_config\_t**

This structure would used to keep the settings for initialization.

#### Data Fields

- bool **enableInDozeMode**

*Control system transition to Stop and Wait power modes while ADC is converting.*
- **lpadc\_conversion\_average\_mode\_t** **conversionAverageMode**

*Auto-Calibration Averages.*
- bool **enableAnalogPreliminary**

*ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).*
- uint32\_t **powerUpDelay**

*When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize.*
- **lpadc\_reference\_voltage\_source\_t** **referenceVoltageSource**

*Selects the voltage reference high used for conversions.*
- **lpadc\_trigger\_priority\_policy\_t** **triggerPriorityPolicy**

*Control how higher priority triggers are handled, see to lpadc\_trigger\_priority\_policy\_t.*
- bool **enableConvPause**

- *Enables the ADC pausing function.*
- **uint32\_t convPauseDelay**  
*Controls the duration of pausing during command execution sequencing.*
- **uint32\_t FIFO0Watermark**  
*FIFO0Watermark is a programmable threshold setting.*
- **uint32\_t FIFO1Watermark**  
*FIFO1 Watermark is a programmable threshold setting.*

## Field Documentation

### (1) **bool Ipadc\_config\_t::enableInDozeMode**

When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

### (2) **Ipadc\_conversion\_average\_mode\_t Ipadc\_config\_t::conversionAverageMode**

### (3) **bool Ipadc\_config\_t::enableAnalogPreliminary**

### (4) **uint32\_t Ipadc\_config\_t::powerUpDelay**

The startup delay count of (powerUpDelay \* 4) ADCK cycles must result in a longer delay than the analog startup time.

### (5) **Ipadc\_reference\_voltage\_source\_t Ipadc\_config\_t::referenceVoltageSource**

### (6) **Ipadc\_trigger\_priority\_policy\_t Ipadc\_config\_t::triggerPriorityPolicy**

### (7) **bool Ipadc\_config\_t::enableConvPause**

When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in "Compare Until True" configuration.

### (8) **uint32\_t Ipadc\_config\_t::convPauseDelay**

The pause delay is a count of (convPauseDelay\*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

### (9) **uint32\_t Ipadc\_config\_t::FIFO0Watermark**

When the number of datawords stored in the ADC Result FIFO0 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

### (10) **uint32\_t Ipadc\_config\_t::FIFO1Watermark**

When the number of datawords stored in the ADC Result FIFO1 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

### 21.3.2 struct lpadc\_conv\_command\_config\_t

#### Data Fields

- `lpadc_sample_scale_mode_t sampleScaleMode`  
*Sample scale mode.*
- `lpadc_sample_scale_mode_t channelBScaleMode`  
*Alternate channel B Scale mode.*
- `lpadc_sample_channel_mode_t sampleChannelMode`  
*Channel sample mode.*
- `uint32_t channelNumber`  
*Channel number, select the channel or channel pair.*
- `uint32_t channelBNumber`  
*Alternate Channel B number, select the channel.*
- `uint32_t chainedNextCommandNumber`  
*Selects the next command to be executed after this command completes.*
- `bool enableAutoChannelIncrement`  
*Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.*
- `uint32_t loopCount`  
*Selects how many times this command executes before finish and transition to the next command or Idle state.*
- `lpadc_hardware_average_mode_t hardwareAverageMode`  
*Hardware average selection.*
- `lpadc_sample_time_mode_t sampleTimeMode`  
*Sample time selection.*
- `lpadc_hardware_compare_mode_t hardwareCompareMode`  
*Hardware compare selection.*
- `uint32_t hardwareCompareValueHigh`  
*Compare Value High.*
- `uint32_t hardwareCompareValueLow`  
*Compare Value Low.*
- `lpadc_conversion_resolution_mode_t conversionResolutionMode`  
*Conversion resolution mode.*
- `bool enableWaitTrigger`  
*Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.*

**Field Documentation**

- (1) `lpadc_sample_scale_mode_t lpadc_conv_command_config_t::sampleScaleMode`
- (2) `lpadc_sample_scale_mode_t lpadc_conv_command_config_t::channelBScaleMode`
- (3) `lpadc_sample_channel_mode_t lpadc_conv_command_config_t::sampleChannelMode`
- (4) `uint32_t lpadc_conv_command_config_t::channelNumber`
- (5) `uint32_t lpadc_conv_command_config_t::channelBNumber`
- (6) `uint32_t lpadc_conv_command_config_t::chainedNextCommandNumber`

1-15 is available, 0 is to terminate the chain after this command.

- (7) `bool lpadc_conv_command_config_t::enableAutoChannelIncrement`
- (8) `uint32_t lpadc_conv_command_config_t::loopCount`

Command executes LOOP+1 times. 0-15 is available.

- (9) `lpadc_hardware_average_mode_t lpadc_conv_command_config_t::hardwareAverageMode`
- (10) `lpadc_sample_time_mode_t lpadc_conv_command_config_t::sampleTimeMode`
- (11) `lpadc_hardware_compare_mode_t lpadc_conv_command_config_t::hardwareCompareMode`
- (12) `uint32_t lpadc_conv_command_config_t::hardwareCompareValueHigh`

The available value range is in 16-bit.

- (13) `uint32_t lpadc_conv_command_config_t::hardwareCompareValueLow`

The available value range is in 16-bit.

- (14) `lpadc_conversion_resolution_mode_t lpadc_conv_command_config_t::conversion-ResolutionMode`
- (15) `bool lpadc_conv_command_config_t::enableWaitTrigger`

### **21.3.3 struct lpadc\_conv\_trigger\_config\_t**

#### **Data Fields**

- `uint32_t targetCommandId`  
*Select the command from command buffer to execute upon detect of the associated trigger event.*
- `uint32_t delayPower`  
*Select the trigger delay duration to wait at the start of servicing a trigger event.*
- `uint32_t priority`

- **bool enableHardwareTrigger**  
*Sets the priority of the associated trigger source.*

*Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not.*

### Field Documentation

- (1) **uint32\_t lpadc\_conv\_trigger\_config\_t::targetCommandId**
- (2) **uint32\_t lpadc\_conv\_trigger\_config\_t::delayPower**

When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is  $2^{\text{delayPower}}$  ADCK cycles. The available value range is 4-bit.

- (3) **uint32\_t lpadc\_conv\_trigger\_config\_t::priority**

If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

- (4) **bool lpadc\_conv\_trigger\_config\_t::enableHardwareTrigger**

The software trigger is always available.

### 21.3.4 struct lpadc\_conv\_result\_t

#### Data Fields

- **uint32\_t commandIdSource**  
*Indicate the command buffer being executed that generated this result.*
- **uint32\_t loopCountIndex**  
*Indicate the loop count value during command execution that generated this result.*
- **uint32\_t triggerIdSource**  
*Indicate the trigger source that initiated a conversion and generated this result.*
- **uint16\_t convValue**  
*Data result.*

### Field Documentation

- (1) **uint32\_t lpadc\_conv\_result\_t::commandIdSource**
- (2) **uint32\_t lpadc\_conv\_result\_t::loopCountIndex**
- (3) **uint32\_t lpadc\_conv\_result\_t::triggerIdSource**
- (4) **uint16\_t lpadc\_conv\_result\_t::convValue**

### 21.3.5 struct lpadc\_calibration\_value\_t

## 21.4 Macro Definition Documentation

**21.4.1 #define FSL\_LPADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 8, 4))**

**21.4.2 #define LPADC\_GET\_ACTIVE\_COMMAND\_STATUS( *statusVal* ) ((*statusVal* & ADC\_STAT\_CMDACT\_MASK) >> ADC\_STAT\_CMDACT\_SHIFT)**

The *statusVal* is the return value from [LPADC\\_GetStatusFlags\(\)](#).

**21.4.3 #define LPADC\_GET\_ACTIVE\_TRIGGER\_STATUS( *statusVal* ) ((*statusVal* & ADC\_STAT\_TRGACT\_MASK) >> ADC\_STAT\_TRGACT\_SHIFT)**

The *statusVal* is the return value from [LPADC\\_GetStatusFlags\(\)](#).

## 21.5 Enumeration Type Documentation

### 21.5.1 enum \_lpadc\_status\_flags

Enumerator

***kLPADC\_ResultFIFO0OverflowFlag*** Indicates that more data has been written to the Result FIFO 0 than it can hold.

***kLPADC\_ResultFIFO0ReadyFlag*** Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

***kLPADC\_ResultFIFO1OverflowFlag*** Indicates that more data has been written to the Result FIFO 1 than it can hold.

***kLPADC\_ResultFIFO1ReadyFlag*** Indicates when the number of valid datawords in the result FIFO 1 is greater than the setting watermark level.

***kLPADC\_TriggerExceptionFlag*** Indicates that a trigger exception event has occurred.

***kLPADC\_TriggerCompletionFlag*** Indicates that a trigger completion event has occurred.

***kLPADC\_CalibrationReadyFlag*** Indicates that the calibration process is done.

***kLPADC\_ActiveFlag*** Indicates that the ADC is in active state.

***kLPADC\_ResultFIFOOverflowFlag*** To compilable with old version, do not recommend using this, please use [kLPADC\\_ResultFIFO0OverflowFlag](#) as instead.

***kLPADC\_ResultFIFOReadyFlag*** To compilable with old version, do not recommend using this, please use [kLPADC\\_ResultFIFO0ReadyFlag](#) as instead.

### 21.5.2 enum \_lpadc\_interrupt\_enable

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

## Enumerator

***kLPADC\_ResultFIFO0OverflowInterruptEnable*** Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

***kLPADC\_FIFO0WatermarkInterruptEnable*** Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

***kLPADC\_ResultFIFOOverflowInterruptEnable*** To compilable with old version, do not recommend using this, please use [\*\*\*kLPADC\\_ResultFIFO0OverflowInterruptEnable\*\*\*](#) as instead.

***kLPADC\_FIFOWatermarkInterruptEnable*** To compilable with old version, do not recommend using this, please use [\*\*\*kLPADC\\_FIFO0WatermarkInterruptEnable\*\*\*](#) as instead.

***kLPADC\_ResultFIFO1OverflowInterruptEnable*** Configures ADC to generate overflow interrupt requests when FOF1 flag is asserted.

***kLPADC\_FIFO1WatermarkInterruptEnable*** Configures ADC to generate watermark interrupt requests when RDY1 flag is asserted.

***kLPADC\_TriggerExceptionInterruptEnable*** Configures ADC to generate trigger exception interrupt.

***kLPADC\_Trigger0CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 0 completion.

***kLPADC\_Trigger1CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 1 completion.

***kLPADC\_Trigger2CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 2 completion.

***kLPADC\_Trigger3CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 3 completion.

***kLPADC\_Trigger4CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 4 completion.

***kLPADC\_Trigger5CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 5 completion.

***kLPADC\_Trigger6CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 6 completion.

***kLPADC\_Trigger7CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 7 completion.

***kLPADC\_Trigger8CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 8 completion.

***kLPADC\_Trigger9CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 9 completion.

***kLPADC\_Trigger10CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 10 completion.

***kLPADC\_Trigger11CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 11 completion.

***kLPADC\_Trigger12CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 12 completion.

***kLPADC\_Trigger13CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 13 completion.

***kLPADC\_Trigger14CompletionInterruptEnable*** Configures ADC to generate interrupt when

trigger 14 completion.

***kLPADC\_Trigger15CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 15 completion.

### 21.5.3 enum \_lpadc\_trigger\_status\_flags

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Enumerator

|                                               |                                                                         |
|-----------------------------------------------|-------------------------------------------------------------------------|
| <b><i>kLPADC_Trigger0InterruptedFlag</i></b>  | Trigger 0 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger1InterruptedFlag</i></b>  | Trigger 1 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger2InterruptedFlag</i></b>  | Trigger 2 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger3InterruptedFlag</i></b>  | Trigger 3 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger4InterruptedFlag</i></b>  | Trigger 4 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger5InterruptedFlag</i></b>  | Trigger 5 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger6InterruptedFlag</i></b>  | Trigger 6 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger7InterruptedFlag</i></b>  | Trigger 7 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger8InterruptedFlag</i></b>  | Trigger 8 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger9InterruptedFlag</i></b>  | Trigger 9 is interrupted by a high priority exception.                  |
| <b><i>kLPADC_Trigger10InterruptedFlag</i></b> | Trigger 10 is interrupted by a high priority exception.                 |
| <b><i>kLPADC_Trigger11InterruptedFlag</i></b> | Trigger 11 is interrupted by a high priority exception.                 |
| <b><i>kLPADC_Trigger12InterruptedFlag</i></b> | Trigger 12 is interrupted by a high priority exception.                 |
| <b><i>kLPADC_Trigger13InterruptedFlag</i></b> | Trigger 13 is interrupted by a high priority exception.                 |
| <b><i>kLPADC_Trigger14InterruptedFlag</i></b> | Trigger 14 is interrupted by a high priority exception.                 |
| <b><i>kLPADC_Trigger15InterruptedFlag</i></b> | Trigger 15 is interrupted by a high priority exception.                 |
| <b><i>kLPADC_Trigger0CompletedFlag</i></b>    | Trigger 0 is completed and trigger 0 has enabled completion interrupts. |
| <b><i>kLPADC_Trigger1CompletedFlag</i></b>    | Trigger 1 is completed and trigger 1 has enabled completion interrupts. |
| <b><i>kLPADC_Trigger2CompletedFlag</i></b>    | Trigger 2 is completed and trigger 2 has enabled completion interrupts. |
| <b><i>kLPADC_Trigger3CompletedFlag</i></b>    | Trigger 3 is completed and trigger 3 has enabled completion interrupts. |
| <b><i>kLPADC_Trigger4CompletedFlag</i></b>    | Trigger 4 is completed and trigger 4 has enabled completion interrupts. |
| <b><i>kLPADC_Trigger5CompletedFlag</i></b>    | Trigger 5 is completed and trigger 5 has enabled completion interrupts. |
| <b><i>kLPADC_Trigger6CompletedFlag</i></b>    | Trigger 6 is completed and trigger 6 has enabled completion interrupts. |
| <b><i>kLPADC_Trigger7CompletedFlag</i></b>    | Trigger 7 is completed and trigger 7 has enabled completion interrupts. |

***kLPADC\_Trigger8CompletedFlag*** Trigger 8 is completed and trigger 8 has enabled completion interrupts.

***kLPADC\_Trigger9CompletedFlag*** Trigger 9 is completed and trigger 9 has enabled completion interrupts.

***kLPADC\_Trigger10CompletedFlag*** Trigger 10 is completed and trigger 10 has enabled completion interrupts.

***kLPADC\_Trigger11CompletedFlag*** Trigger 11 is completed and trigger 11 has enabled completion interrupts.

***kLPADC\_Trigger12CompletedFlag*** Trigger 12 is completed and trigger 12 has enabled completion interrupts.

***kLPADC\_Trigger13CompletedFlag*** Trigger 13 is completed and trigger 13 has enabled completion interrupts.

***kLPADC\_Trigger14CompletedFlag*** Trigger 14 is completed and trigger 14 has enabled completion interrupts.

***kLPADC\_Trigger15CompletedFlag*** Trigger 15 is completed and trigger 15 has enabled completion interrupts.

#### 21.5.4 enum lpadc\_sample\_scale\_mode\_t

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Enumerator

***kLPADC\_SamplePartScale*** Use divided input voltage signal. (For scale select,please refer to the reference manual).

***kLPADC\_SampleFullScale*** Full scale (Factor of 1).

#### 21.5.5 enum lpadc\_sample\_channel\_mode\_t

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Enumerator

***kLPADC\_SampleChannelSingleEndSideA*** Single-end mode, only A-side channel is converted.

***kLPADC\_SampleChannelSingleEndSideB*** Single-end mode, only B-side channel is converted.

***kLPADC\_SampleChannelDiffBothSide*** Differential mode, the ADC result is (CHnA-CHnB).

***kLPADC\_SampleChannelDualSingleEndBothSide*** Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

## 21.5.6 enum lpadc.hardware\_average\_mode\_t

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note

Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Enumerator

- kLPADC\_HardwareAverageCount1*** Single conversion.
- kLPADC\_HardwareAverageCount2*** 2 conversions averaged.
- kLPADC\_HardwareAverageCount4*** 4 conversions averaged.
- kLPADC\_HardwareAverageCount8*** 8 conversions averaged.
- kLPADC\_HardwareAverageCount16*** 16 conversions averaged.
- kLPADC\_HardwareAverageCount32*** 32 conversions averaged.
- kLPADC\_HardwareAverageCount64*** 64 conversions averaged.
- kLPADC\_HardwareAverageCount128*** 128 conversions averaged.
- kLPADC\_HardwareAverageCount256*** 256 conversions averaged.
- kLPADC\_HardwareAverageCount512*** 512 conversions averaged.
- kLPADC\_HardwareAverageCount1024*** 1024 conversions averaged.

## 21.5.7 enum lpadc\_sample\_time\_mode\_t

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Enumerator

- kLPADC\_SampleTimeADCK3*** 3 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK5*** 5 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK7*** 7 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK11*** 11 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK19*** 19 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK35*** 35 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK67*** 69 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK131*** 131 ADCK cycles total sample time.

### 21.5.8 enum lpadc\_hardware\_compare\_mode\_t

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Enumerator

***kLPADC\_HardwareCompareDisabled*** Compare disabled.

***kLPADC\_HardwareCompareStoreOnTrue*** Compare enabled. Store on true.

***kLPADC\_HardwareCompareRepeatUntilTrue*** Compare enabled. Repeat channel acquisition until true.

### 21.5.9 enum lpadc\_conversion\_resolution\_mode\_t

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to [lpadc\\_sample\\_channel\\_mode\\_t](#)

Enumerator

***kLPADC\_ConversionResolutionStandard*** Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

***kLPADC\_ConversionResolutionHigh*** High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

### 21.5.10 enum lpadc\_conversion\_average\_mode\_t

Configure the converion average number for auto-calibration.

Note

Some enumerator values are not available on some devices, mainly depends on the size of CAL\_A-VGS field in CTRL register.

Enumerator

***kLPADC\_ConversionAverage1*** Single conversion.

***kLPADC\_ConversionAverage2*** 2 conversions averaged.

***kLPADC\_ConversionAverage4*** 4 conversions averaged.

***kLPADC\_ConversionAverage8*** 8 conversions averaged.

***kLPADC\_ConversionAverage16*** 16 conversions averaged.

***kLPADC\_ConversionAverage32*** 32 conversions averaged.

***kLPADC\_ConversionAverage64*** 64 conversions averaged.

*kLPADC\_ConversionAverage128* 128 conversions averaged.  
*kLPADC\_ConversionAverage256* 256 conversions averaged.  
*kLPADC\_ConversionAverage512* 512 conversions averaged.  
*kLPADC\_ConversionAverage1024* 1024 conversions averaged.

### 21.5.11 enum lpadc\_reference\_voltage\_source\_t

For detail information, need to check the SoC's specification.

Enumerator

*kLPADC\_ReferenceVoltageAlt1* Option 1 setting.  
*kLPADC\_ReferenceVoltageAlt2* Option 2 setting.  
*kLPADC\_ReferenceVoltageAlt3* Option 3 setting.

### 21.5.12 enum lpadc\_power\_level\_mode\_t

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Enumerator

*kLPADC\_PowerLevelAlt1* Lowest power setting.  
*kLPADC\_PowerLevelAlt2* Next lowest power setting.  
*kLPADC\_PowerLevelAlt3* ...  
*kLPADC\_PowerLevelAlt4* Highest power setting.

### 21.5.13 enum lpadc\_offset\_calibration\_mode\_t

Enumerator

*kLPADC\_OffsetCalibration12bitMode* 12 bit offset calibration mode.  
*kLPADC\_OffsetCalibration16bitMode* 16 bit offset calibration mode.

### 21.5.14 enum lpadc\_trigger\_priority\_policy\_t

This selection controls how higher priority triggers are handled.

## Note

**kLPADC\_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

## Enumerator

***kLPADC\_ConvPreemptImmediatelyNotAutoResumed*** If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

***kLPADC\_ConvPreemptSoftlyNotAutoResumed*** If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

***kLPADC\_ConvPreemptImmediatelyAutoRestarted*** If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

***kLPADC\_ConvPreemptSoftlyAutoRestarted*** If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

***kLPADC\_ConvPreemptImmediatelyAutoResumed*** If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

***kLPADC\_ConvPreemptSoftlyAutoResumed*** If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

***kLPADC\_TriggerPriorityPreemptImmediately*** Legacy support is not recommended as it only ensures compatibility with older versions.

***kLPADC\_TriggerPriorityPreemptSoftly*** Legacy support is not recommended as it only ensures compatibility with older versions.

***kLPADC\_ConvPreemptSubsequentlyNotAutoResumed*** If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger, when higher priority conversion finishes, the preempted conversion will not automatically be restarted or resumed.

***kLPADC\_ConvPreemptSubsequentlyAutoRestarted*** If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger, when higher priority conversion finishes, the preempted conversion will be automatically restarted.

***kLPADC\_ConvPreemptSubsequentlyAutoResumed*** If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger, when higher priority conversion finishes, the preempted conversion will be automatically resumed.

***kLPADC\_TriggerPriorityPreemptSubsequently*** Legacy support is not recommended as it only ensures compatibility with older versions.

***kLPADC\_TriggerPriorityExceptionDisabled*** High priority trigger exception disabled.

## 21.6 Function Documentation

### 21.6.1 void LPADC\_Init ( ADC\_Type \* *base*, const lpadc\_config\_t \* *config* )

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | LPADC peripheral base address.                            |
| <i>config</i> | Pointer to configuration structure. See "lpadc_config_t". |

### 21.6.2 void LPADC\_GetDefaultConfig ( lpadc\_config\_t \* *config* )

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

```
* config->enableInDozeMode = true;
* config->enableAnalogPreliminary = false;
* config->powerUpDelay = 0x80;
* config->referenceVoltageSource = kLPADC_ReferenceVoltageAlt1;
* config->powerLevelMode = kLPADC_PowerLevelAlt1;
* config->triggerPriorityPolicy = kLPADC_TriggerPriorityPreemptImmediately
;
* config->enableConvPause = false;
* config->convPauseDelay = 0U;
* config->FIFOWatermark = 0U;
*
```

Parameters

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

### 21.6.3 void LPADC\_Deinit ( ADC\_Type \* *base* )

Parameters

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

#### 21.6.4 static void LPADC\_Enable( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | switcher to the module.        |

#### 21.6.5 static void LPADC\_DoResetFIFO0( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

#### 21.6.6 static void LPADC\_DoResetFIFO1( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

#### 21.6.7 static void LPADC\_DoResetConfig( ADC\_Type \* *base* ) [inline], [static]

Reset all ADC internal logic and registers, except the Control Register (ADCx\_CTRL).

Parameters

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

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

Parameters

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

Returns

status flags' mask. See to [\\_lpadc\\_status\\_flags](#).

### 21.6.9 static void LPADC\_ClearStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Only the flags can be cleared by writing ADCx\_STATUS register would be cleared by this API.

Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                   |
| <i>mask</i> | Mask value for flags to be cleared. See to <a href="#">_lpadc_status_flags</a> . |

### 21.6.10 static uint32\_t LPADC\_GetTriggerStatusFlags ( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The OR'ed value of [\\_lpadc\\_trigger\\_status\\_flags](#).

### 21.6.11 static void LPADC\_ClearTriggerStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                                                             |
| <i>mask</i> | The mask of trigger status flags to be cleared, should be the OR'ed value of <a href="#">_lpadc_trigger_status_flags</a> . |

**21.6.12 static void LPADC\_EnableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                    |
| <i>mask</i> | Mask value for interrupt events. See to <a href="#">_lpadc_interrupt_enable</a> . |

**21.6.13 static void LPADC\_DisableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                    |
| <i>mask</i> | Mask value for interrupt events. See to <a href="#">_lpadc_interrupt_enable</a> . |

**21.6.14 static void LPADC\_EnableFIFO0WatermarkDMA ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | Switcher to the event.         |

**21.6.15 static void LPADC\_EnableFIFO1WatermarkDMA ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | Switcher to the event.         |

### 21.6.16 static uint32\_t LPADC\_GetConvResultCount ( ADC\_Type \* *base*, uint8\_t *index* ) [inline], [static]

Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | LPADC peripheral base address. |
| <i>index</i> | Result FIFO index.             |

Returns

The count of result kept in conversion FIFO.

### 21.6.17 bool LPADC\_GetConvResult ( ADC\_Type \* *base*, lpadc\_conv\_result\_t \* *result*, uint8\_t *index* )

Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | LPADC peripheral base address.                                                     |
| <i>result</i> | Pointer to structure variable that keeps the conversion result in conversion FIFO. |
| <i>index</i>  | Result FIFO index.                                                                 |

Returns

Status whether FIFO entry is valid.

### 21.6.18 void LPADC\_GetConvResultBlocking ( ADC\_Type \* *base*, lpadc\_conv\_result\_t \* *result*, uint8\_t *index* )

Parameters

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <i>base</i>   | LPADC peripheral base address.                                                       |
| <i>result</i> | Pointer to structure variable that keeps the conversion result in conversion FIFOOn. |
| <i>index</i>  | Result FIFO index.                                                                   |

### 21.6.19 void LPADC\_SetConvTriggerConfig ( ADC\_Type \* *base*, uint32\_t *triggerId*, const lpadc\_conv\_trigger\_config\_t \* *config* )

Each programmable trigger can launch the conversion command in command buffer.

Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>base</i>      | LPADC peripheral base address.                                                           |
| <i>triggerId</i> | ID for each trigger. Typically, the available value range is from 0.                     |
| <i>config</i>    | Pointer to configuration structure. See to <a href="#">lpadc_conv_trigger_config_t</a> . |

### 21.6.20 void LPADC\_GetDefaultConvTriggerConfig ( lpadc\_conv\_trigger\_config\_t \* *config* )

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
* config->targetCommandId = 0U;
* config->delayPower = 0U;
* config->priority = 0U;
* config->channelAFIFOSelect = 0U;
* config->channelBFIFOSelect = 0U;
* config->enableHardwareTrigger = false;
*
```

Parameters

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

### 21.6.21 static void LPADC\_DoSoftwareTrigger ( ADC\_Type \* *base*, uint32\_t *triggerIdMask* ) [inline], [static]

## Parameters

|                      |                                                                 |
|----------------------|-----------------------------------------------------------------|
| <i>base</i>          | LPADC peripheral base address.                                  |
| <i>triggerIdMask</i> | Mask value for software trigger indexes, which count from zero. |

**21.6.22 void LPADC\_SetConvCommandConfig ( ADC\_Type \* *base*, uint32\_t *commandId*, const lpadc\_conv\_command\_config\_t \* *config* )**

## Note

The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

## Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>base</i>      | LPADC peripheral base address.                                                           |
| <i>commandId</i> | ID for command in command buffer. Typically, the available value range is 1 - 15.        |
| <i>config</i>    | Pointer to configuration structure. See to <a href="#">lpadc_conv_command_config_t</a> . |

**21.6.23 void LPADC\_GetDefaultConvCommandConfig ( lpadc\_conv\_command\_config\_t \* *config* )**

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```

* config->sampleScaleMode = kLPADC_SampleFullScale;
* config->channelBScaleMode = kLPADC_SampleFullScale;
* config->sampleChannelMode = kLPADC_SampleChannelSingleEndSideA
;
* config->channelNumber = OU;
* config->channelBNumber = OU;
* config->chainedNextCommandNumber = OU;
* config->enableAutoChannelIncrement = false;
* config->loopCount = OU;
* config->hardwareAverageMode = kLPADC_HardwareAverageCount1;
* config->sampleTimeMode = kLPADC_SampleTimeADCK3;
* config->hardwareCompareMode = kLPADC_HardwareCompareDisabled;
* config->hardwareCompareValueHigh = OU;
* config->hardwareCompareValueLow = OU;
* config->conversionResolutionMode = kLPADC_ConversionResolutionStandard
;
* config->enableWaitTrigger = false;
* config->enableChannelB = false;
*
```

Parameters

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

#### 21.6.24 static void LPADC\_SetOffset12BitValue ( ADC\_Type \* *base*, uint32\_t *valueA*, uint32\_t *valueB* ) [inline], [static]

Set the offset trim value for offset calibration manually.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>valueA</i> | Setting offset value A.        |
| <i>valueB</i> | Setting offset value B.        |

Note

In normal adc sequence, the values are automatically calculated by LPADC\_EnableOffsetCalibration.

#### 21.6.25 static void LPADC\_SetOffset16BitValue ( ADC\_Type \* *base*, uint32\_t *valueA*, uint32\_t *valueB* ) [inline], [static]

Set the offset trim value for offset calibration manually.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>valueA</i> | Setting offset value A.        |
| <i>valueB</i> | Setting offset value B.        |

Note

In normal adc sequence, the values are automatically calculated by LPADC\_EnableOffsetCalibration.

#### 21.6.26 static void LPADC\_EnableOffsetCalibration ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | LPADC peripheral base address.        |
| <i>enable</i> | switcher to the calibration function. |

### 21.6.27 static void LPADC\_SetOffsetCalibrationMode ( ADC\_Type \* *base*, lpadc\_offset\_calibration\_mode\_t *mode* ) [inline], [static]

Parameters

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                       |
| <i>mode</i> | set offset calibration mode.see to <a href="#">lpadc_offset_calibration_mode_t</a> . |

### 21.6.28 void LPADC\_DoOffsetCalibration ( ADC\_Type \* *base* )

Parameters

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

### 21.6.29 void LPADC\_DoAutoCalibration ( ADC\_Type \* *base* )

Parameters

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

### 21.6.30 void LPADC\_PrepAutoCalibration ( ADC\_Type \* *base* )

LPADC\_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

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

### 21.6.31 void LPADC\_FinishAutoCalibration ( ADC\_Type \* *base* )

Parameters

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

### 21.6.32 void LPADC\_GetCalibrationValue ( ADC\_Type \* *base*, lpadc\_calibration\_value\_t \* *ptrCalibrationValue* )

Note

Please note the ADC will be disabled temporary.  
This function should be used after finish calibration.

Parameters

|                             |                                                                                                                                        |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>                 | LPADC peripheral base address.                                                                                                         |
| <i>ptrCalibration-Value</i> | Pointer to <a href="#">lpadc_calibration_value_t</a> structure, this memory block should be always powered on even in low power modes. |

### 21.6.33 void LPADC\_SetCalibrationValue ( ADC\_Type \* *base*, const lpadc\_calibration\_value\_t \* *ptrCalibrationValue* )

Note

Please note the ADC will be disabled temporary.

Parameters

|                             |                                                                                                        |
|-----------------------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>                 | LPADC peripheral base address.                                                                         |
| <i>ptrCalibration-Value</i> | Pointer to <a href="#">lpadc_calibration_value_t</a> structure which contains ADC's calibration value. |

# Chapter 22

## LPI2C: Low Power Inter-Integrated Circuit Driver

### 22.1 Overview

#### Modules

- LPI2C CMSIS Driver
- LPI2C FreeRTOS Driver
- LPI2C Master DMA Driver
- LPI2C Master Driver
- LPI2C Slave Driver

#### Macros

- #define I2C\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*

#### Enumerations

- enum {  
    kStatus\_LPI2C\_Busy = MAKE\_STATUS(kStatusGroup\_LPI2C, 0),  
    kStatus\_LPI2C\_Idle = MAKE\_STATUS(kStatusGroup\_LPI2C, 1),  
    kStatus\_LPI2C\_Nak = MAKE\_STATUS(kStatusGroup\_LPI2C, 2),  
    kStatus\_LPI2C\_FifoError = MAKE\_STATUS(kStatusGroup\_LPI2C, 3),  
    kStatus\_LPI2C\_BitError = MAKE\_STATUS(kStatusGroup\_LPI2C, 4),  
    kStatus\_LPI2C\_ArbitrationLost = MAKE\_STATUS(kStatusGroup\_LPI2C, 5),  
    kStatus\_LPI2C\_PinLowTimeout,  
    kStatus\_LPI2C\_NoTransferInProgress,  
    kStatus\_LPI2C\_DmaRequestFail = MAKE\_STATUS(kStatusGroup\_LPI2C, 8),  
    kStatus\_LPI2C\_Timeout = MAKE\_STATUS(kStatusGroup\_LPI2C, 9) }  
*LPI2C status return codes.*

#### Functions

- uint32\_t LPI2C\_GetInstance (LPI2C\_Type \*base)  
*Returns an instance number given a base address.*

#### Variables

- IRQn\_Type const kLpi2cIrqs []  
*Array to map LPI2C instance number to IRQ number, used internally for LPI2C master int APIs.*
- lpi2c\_master\_isr\_t s\_lpi2cMasterIsr

- Pointer to master IRQ handler for each instance, used internally for LPI2C master *APIs*.
- `void * s_lpi2cMasterHandle []`  
Pointers to master handles for each instance, used internally for LPI2C master interrupt *APIs*.

## Driver version

- `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))`  
*LPI2C driver version.*

## 22.2 Macro Definition Documentation

### 22.2.1 `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))`

### 22.2.2 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

## 22.3 Enumeration Type Documentation

### 22.3.1 anonymous enum

Enumerator

`kStatus_LPI2C_Busy` The master is already performing a transfer.

`kStatus_LPI2C_Idle` The slave driver is idle.

`kStatus_LPI2C_Nak` The slave device sent a NAK in response to a byte.

`kStatus_LPI2C_FifoError` FIFO under run or overrun.

`kStatus_LPI2C_BitError` Transferred bit was not seen on the bus.

`kStatus_LPI2C_ArbitrationLost` Arbitration lost error.

`kStatus_LPI2C_PinLowTimeout` SCL or SDA were held low longer than the timeout.

`kStatus_LPI2C_NoTransferInProgress` Attempt to abort a transfer when one is not in progress.

`kStatus_LPI2C_DmaRequestFail` DMA request failed.

`kStatus_LPI2C_Timeout` Timeout polling status flags.

## 22.4 Function Documentation

### 22.4.1 `uint32_t LPI2CGetInstance ( LPI2C_Type * base )`

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

LPI2C instance number starting from 0.

## 22.5 Variable Documentation

### 22.5.1 IRQn\_Type const kLpi2cIrqs[]

### 22.5.2 lpi2c\_master\_isr\_t s\_lpi2cMasterIsr

### 22.5.3 void\* s\_lpi2cMasterHandle[]

## 22.6 LPI2C Master Driver

### 22.6.1 Overview

#### Data Structures

- struct `lpi2c_master_config_t`  
*Structure with settings to initialize the LPI2C master module. [More...](#)*
- struct `lpi2c_data_match_config_t`  
*LPI2C master data match configuration structure. [More...](#)*
- struct `lpi2c_master_transfer_t`  
*Non-blocking transfer descriptor structure. [More...](#)*
- struct `lpi2c_master_handle_t`  
*Driver handle for master non-blocking APIs. [More...](#)*

#### Typedefs

- typedef void(\* `lpi2c_master_transfer_callback_t` )(LPI2C\_Type \*base, `lpi2c_master_handle_t` \*handle, `status_t` completionStatus, void \*userData)  
*Master completion callback function pointer type.*
- typedef void(\* `lpi2c_master_isr_t` )(LPI2C\_Type \*base, void \*handle)  
*Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.*

#### Enumerations

- enum `_lpi2c_master_flags` {
   
`kLPI2C_MasterTxReadyFlag` = LPI2C\_MSR\_TDF\_MASK,  
`kLPI2C_MasterRxReadyFlag` = LPI2C\_MSR\_RDF\_MASK,  
`kLPI2C_MasterEndOfPacketFlag` = LPI2C\_MSR\_EPF\_MASK,  
`kLPI2C_MasterStopDetectFlag` = LPI2C\_MSR\_SDF\_MASK,  
`kLPI2C_MasterNackDetectFlag` = LPI2C\_MSR\_NDF\_MASK,  
`kLPI2C_MasterArbitrationLostFlag` = LPI2C\_MSR\_ALF\_MASK,  
`kLPI2C_MasterFifoErrFlag` = LPI2C\_MSR\_FEF\_MASK,  
`kLPI2C_MasterPinLowTimeoutFlag` = LPI2C\_MSR\_PLTF\_MASK,  
`kLPI2C_MasterDataMatchFlag` = LPI2C\_MSR\_DMF\_MASK,  
`kLPI2C_MasterBusyFlag` = LPI2C\_MSR\_MBF\_MASK,  
`kLPI2C_MasterBusBusyFlag` = LPI2C\_MSR\_BBF\_MASK,  
`kLPI2C_MasterClearFlags`,  
`kLPI2C_MasterIrqFlags`,  
`kLPI2C_MasterErrorFlags` }
   
*LPI2C master peripheral flags.*
- enum `lpi2c_direction_t` {
   
`kLPI2C_Write` = 0U,  
`kLPI2C_Read` = 1U }

- *Direction of master and slave transfers.*
- enum `lpi2c_master_pin_config_t` {
   
  `kLPI2C_2PinOpenDrain` = 0x0U,
   
  `kLPI2C_2PinOutputOnly` = 0x1U,
   
  `kLPI2C_2PinPushPull` = 0x2U,
   
  `kLPI2C_4PinPushPull` = 0x3U,
   
  `kLPI2C_2PinOpenDrainWithSeparateSlave`,
   
  `kLPI2C_2PinOutputOnlyWithSeparateSlave`,
   
  `kLPI2C_2PinPushPullWithSeparateSlave`,
   
  `kLPI2C_4PinPushPullWithInvertedOutput` = 0x7U }
- LPI2C pin configuration.*
- enum `lpi2c_host_request_source_t` {
   
  `kLPI2C_HostRequestExternalPin` = 0x0U,
   
  `kLPI2C_HostRequestInputTrigger` = 0x1U }
- LPI2C master host request selection.*
- enum `lpi2c_host_request_polarity_t` {
   
  `kLPI2C_HostRequestPinActiveLow` = 0x0U,
   
  `kLPI2C_HostRequestPinActiveHigh` = 0x1U }
- LPI2C master host request pin polarity configuration.*
- enum `lpi2c_data_match_config_mode_t` {
   
  `kLPI2C_MatchDisabled` = 0x0U,
   
  `kLPI2C_1stWordEqualsM0OrM1` = 0x2U,
   
  `kLPI2C_AnyWordEqualsM0OrM1` = 0x3U,
   
  `kLPI2C_1stWordEqualsM0And2ndWordEqualsM1`,
   
  `kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1`,
   
  `kLPI2C_1stWordAndM1EqualsM0AndM1`,
   
  `kLPI2C_AnyWordAndM1EqualsM0AndM1` }
- LPI2C master data match configuration modes.*
- enum `_lpi2c_master_transfer_flags` {
   
  `kLPI2C_TransferDefaultFlag` = 0x00U,
   
  `kLPI2C_TransferNoStartFlag` = 0x01U,
   
  `kLPI2C_TransferRepeatedStartFlag` = 0x02U,
   
  `kLPI2C_TransferNoStopFlag` = 0x04U }
- Transfer option flags.*

## Initialization and deinitialization

- void `LPI2C_MasterGetDefaultConfig` (`lpi2c_master_config_t` \*`masterConfig`)
   
    *Provides a default configuration for the LPI2C master peripheral.*
- void `LPI2C_MasterInit` (`LPI2C_Type` \*`base`, const `lpi2c_master_config_t` \*`masterConfig`, `uint32_t` `sourceClock_Hz`)
   
    *Initializes the LPI2C master peripheral.*
- void `LPI2C_MasterDeinit` (`LPI2C_Type` \*`base`)
   
    *Deinitializes the LPI2C master peripheral.*
- void `LPI2C_MasterConfigureDataMatch` (`LPI2C_Type` \*`base`, const `lpi2c_data_match_config_t` \*`matchConfig`)

- **status\_t LPI2C\_MasterCheckAndClearError** (LPI2C\_Type \*base, uint32\_t status)
 

*Convert provided flags to status code, and clear any errors if present.*
- **status\_t LPI2C\_CheckForBusyBus** (LPI2C\_Type \*base)
 

*Make sure the bus isn't already busy.*
- **static void LPI2C\_MasterReset** (LPI2C\_Type \*base)
 

*Performs a software reset.*
- **static void LPI2C\_MasterEnable** (LPI2C\_Type \*base, bool enable)
 

*Enables or disables the LPI2C module as master.*

## Status

- **static uint32\_t LPI2C\_MasterGetStatusFlags** (LPI2C\_Type \*base)
 

*Gets the LPI2C master status flags.*
- **static void LPI2C\_MasterClearStatusFlags** (LPI2C\_Type \*base, uint32\_t statusMask)
 

*Clears the LPI2C master status flag state.*

## Interrupts

- **static void LPI2C\_MasterEnableInterrupts** (LPI2C\_Type \*base, uint32\_t interruptMask)
 

*Enables the LPI2C master interrupt requests.*
- **static void LPI2C\_MasterDisableInterrupts** (LPI2C\_Type \*base, uint32\_t interruptMask)
 

*Disables the LPI2C master interrupt requests.*
- **static uint32\_t LPI2C\_MasterGetEnabledInterrupts** (LPI2C\_Type \*base)
 

*Returns the set of currently enabled LPI2C master interrupt requests.*

## DMA control

- **static void LPI2C\_MasterEnableDMA** (LPI2C\_Type \*base, bool enableTx, bool enableRx)
 

*Enables or disables LPI2C master DMA requests.*
- **static uint32\_t LPI2C\_MasterGetTxFifoAddress** (LPI2C\_Type \*base)
 

*Gets LPI2C master transmit data register address for DMA transfer.*
- **static uint32\_t LPI2C\_MasterGetRxFifoAddress** (LPI2C\_Type \*base)
 

*Gets LPI2C master receive data register address for DMA transfer.*

## FIFO control

- **static void LPI2C\_MasterSetWatermarks** (LPI2C\_Type \*base, size\_t txWords, size\_t rxWords)
 

*Sets the watermarks for LPI2C master FIFOs.*
- **static void LPI2C\_MasterGetFifoCounts** (LPI2C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)
 

*Gets the current number of words in the LPI2C master FIFOs.*

## Bus operations

- void [LPI2C\\_MasterSetBaudRate](#) (LPI2C\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t baudRate\_Hz)  
*Sets the I2C bus frequency for master transactions.*
- static bool [LPI2C\\_MasterGetBusIdleState](#) (LPI2C\_Type \*base)  
*Returns whether the bus is idle.*
- [status\\_t LPI2C\\_MasterStart](#) (LPI2C\_Type \*base, uint8\_t address, lpi2c\_direction\_t dir)  
*Sends a START signal and slave address on the I2C bus.*
- static [status\\_t LPI2C\\_MasterRepeatedStart](#) (LPI2C\_Type \*base, uint8\_t address, lpi2c\_direction\_t dir)  
*Sends a repeated START signal and slave address on the I2C bus.*
- [status\\_t LPI2C\\_MasterSend](#) (LPI2C\_Type \*base, void \*txBuff, size\_t txSize)  
*Performs a polling send transfer on the I2C bus.*
- [status\\_t LPI2C\\_MasterReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize)  
*Performs a polling receive transfer on the I2C bus.*
- [status\\_t LPI2C\\_MasterStop](#) (LPI2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- [status\\_t LPI2C\\_MasterTransferBlocking](#) (LPI2C\_Type \*base, lpi2c\_master\_transfer\_t \*transfer)  
*Performs a master polling transfer on the I2C bus.*

## Non-blocking

- void [LPI2C\\_MasterTransferCreateHandle](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle, lpi2c\_master\_transfer\_callback\_t callback, void \*userData)  
*Creates a new handle for the LPI2C master non-blocking APIs.*
- [status\\_t LPI2C\\_MasterTransferNonBlocking](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle, lpi2c\_master\_transfer\_t \*transfer)  
*Performs a non-blocking transaction on the I2C bus.*
- [status\\_t LPI2C\\_MasterTransferGetCount](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- void [LPI2C\\_MasterTransferAbort](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle)  
*Terminates a non-blocking LPI2C master transmission early.*

## IRQ handler

- void [LPI2C\\_MasterTransferHandleIRQ](#) (LPI2C\_Type \*base, void \*lpi2cMasterHandle)  
*Reusable routine to handle master interrupts.*

## 22.6.2 Data Structure Documentation

### 22.6.2.1 struct lpi2c\_master\_config\_t

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_MasterGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

#### Data Fields

- bool `enableMaster`  
*Whether to enable master mode.*
- bool `enableDoze`  
*Whether master is enabled in doze mode.*
- bool `debugEnable`  
*Enable transfers to continue when halted in debug mode.*
- bool `ignoreAck`  
*Whether to ignore ACK/NACK.*
- `lpi2c_master_pin_config_t pinConfig`  
*The pin configuration option.*
- `uint32_t baudRate_Hz`  
*Desired baud rate in Hertz.*
- `uint32_t busIdleTimeout_ns`  
*Bus idle timeout in nanoseconds.*
- `uint32_t pinLowTimeout_ns`  
*Pin low timeout in nanoseconds.*
- `uint8_t sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SDA pin.*
- `uint8_t sclGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SCL pin.*
- struct {
  - bool `enable`  
*Enable host request.*
  - `lpi2c_host_request_source_t source`  
*Host request source.*
  - `lpi2c_host_request_polarity_t polarity`  
*Host request pin polarity.*
} `hostRequest`  
*Host request options.*

**Field Documentation**

- (1) **bool lpi2c\_master\_config\_t::enableMaster**
- (2) **bool lpi2c\_master\_config\_t::enableDoze**
- (3) **bool lpi2c\_master\_config\_t::debugEnable**
- (4) **bool lpi2c\_master\_config\_t::ignoreAck**
- (5) **lpi2c\_master\_pin\_config\_t lpi2c\_master\_config\_t::pinConfig**
- (6) **uint32\_t lpi2c\_master\_config\_t::baudRate\_Hz**
- (7) **uint32\_t lpi2c\_master\_config\_t::busIdleTimeout\_ns**

Set to 0 to disable.

- (8) **uint32\_t lpi2c\_master\_config\_t::pinLowTimeout\_ns**

Set to 0 to disable.

- (9) **uint8\_t lpi2c\_master\_config\_t::sdaGlitchFilterWidth\_ns**

Set to 0 to disable.

- (10) **uint8\_t lpi2c\_master\_config\_t::sclGlitchFilterWidth\_ns**

Set to 0 to disable.

- (11) **bool lpi2c\_master\_config\_t::enable**

- (12) **lpi2c\_host\_request\_source\_t lpi2c\_master\_config\_t::source**

- (13) **lpi2c\_host\_request\_polarity\_t lpi2c\_master\_config\_t::polarity**

- (14) **struct { ... } lpi2c\_master\_config\_t::hostRequest**

### **22.6.2.2 struct lpi2c\_data\_match\_config\_t**

#### **Data Fields**

- **lpi2c\_data\_match\_config\_mode\_t matchMode**  
*Data match configuration setting.*
- **bool rxDataMatchOnly**  
*When set to true, received data is ignored until a successful match.*
- **uint32\_t match0**  
*Match value 0.*
- **uint32\_t match1**  
*Match value 1.*

## Field Documentation

- (1) `lpi2c_data_match_config_mode_t lpi2c_data_match_config_t::matchMode`
- (2) `bool lpi2c_data_match_config_t::rxDataMatchOnly`
- (3) `uint32_t lpi2c_data_match_config_t::match0`
- (4) `uint32_t lpi2c_data_match_config_t::match1`

### 22.6.2.3 struct \_lpi2c\_master\_transfer

LPI2C master descriptor of the transfer.

This structure is used to pass transaction parameters to the [LPI2C\\_MasterTransferNonBlocking\(\)](#) API.

## Data Fields

- `uint32_t flags`  
*Bit mask of options for the transfer.*
- `uint16_t slaveAddress`  
*The 7-bit slave address.*
- `lpi2c_direction_t direction`  
*Either `kLPI2C_Read` or `kLPI2C_Write`.*
- `uint32_t subaddress`  
*Sub address.*
- `size_t subaddressSize`  
*Length of sub address to send in bytes.*
- `void * data`  
*Pointer to data to transfer.*
- `size_t dataSize`  
*Number of bytes to transfer.*

## Field Documentation

- (1) `uint32_t lpi2c_master_transfer_t::flags`

See enumeration [\\_lpi2c\\_master\\_transfer\\_flags](#) for available options. Set to 0 or [kLPI2C\\_TransferDefault-Flag](#) for normal transfers.

- (2) `uint16_t lpi2c_master_transfer_t::slaveAddress`
- (3) `lpi2c_direction_t lpi2c_master_transfer_t::direction`
- (4) `uint32_t lpi2c_master_transfer_t::subaddress`

Transferred MSB first.

**(5) size\_t lpi2c\_master\_transfer\_t::subaddressSize**

Maximum size is 4 bytes.

**(6) void\* lpi2c\_master\_transfer\_t::data****(7) size\_t lpi2c\_master\_transfer\_t::dataSize****22.6.2.4 struct \_lpi2c\_master\_handle**

LPI2C master handle of the transfer.

Note

The contents of this structure are private and subject to change.

**Data Fields**

- uint8\_t **state**  
*Transfer state machine current state.*
- uint16\_t **remainingBytes**  
*Remaining byte count in current state.*
- uint8\_t \* **buf**  
*Buffer pointer for current state.*
- uint16\_t **commandBuffer [6]**  
*LPI2C command sequence.*
- lpi2c\_master\_transfer\_t **transfer**  
*Copy of the current transfer info.*
- lpi2c\_master\_transfer\_callback\_t **completionCallback**  
*Callback function pointer.*
- void \* **userData**  
*Application data passed to callback.*

**Field Documentation****(1) uint8\_t lpi2c\_master\_handle\_t::state****(2) uint16\_t lpi2c\_master\_handle\_t::remainingBytes****(3) uint8\_t\* lpi2c\_master\_handle\_t::buf****(4) uint16\_t lpi2c\_master\_handle\_t::commandBuffer[6]**

When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

- (5) `lpi2c_master_transfer_t lpi2c_master_handle_t::transfer`
- (6) `lpi2c_master_transfer_callback_t lpi2c_master_handle_t::completionCallback`
- (7) `void* lpi2c_master_handle_t::userData`

### 22.6.3 Typedef Documentation

#### 22.6.3.1 `typedef void(* lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)`

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [LPI2C\\_MasterTransferCreateHandle\(\)](#).

## Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <i>base</i>              | The LPI2C peripheral base address.                                             |
| <i>handle</i>            | Pointer to the LPI2C master driver handle.                                     |
| <i>completion-Status</i> | Either kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                     |

**22.6.4 Enumeration Type Documentation****22.6.4.1 enum \_lpi2c\_master\_flags**

The following status register flags can be cleared:

- [kLPI2C\\_MasterEndOfPacketFlag](#)
- [kLPI2C\\_MasterStopDetectFlag](#)
- [kLPI2C\\_MasterNackDetectFlag](#)
- [kLPI2C\\_MasterArbitrationLostFlag](#)
- [kLPI2C\\_MasterFifoErrFlag](#)
- [kLPI2C\\_MasterPinLowTimeoutFlag](#)
- [kLPI2C\\_MasterDataMatchFlag](#)

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be enabled as interrupts.

## Note

These enums are meant to be OR'd together to form a bit mask.

## Enumerator

- kLPI2C\_MasterTxReadyFlag* Transmit data flag.  
*kLPI2C\_MasterRxReadyFlag* Receive data flag.  
*kLPI2C\_MasterEndOfPacketFlag* End Packet flag.  
*kLPI2C\_MasterStopDetectFlag* Stop detect flag.  
*kLPI2C\_MasterNackDetectFlag* NACK detect flag.  
*kLPI2C\_MasterArbitrationLostFlag* Arbitration lost flag.  
*kLPI2C\_MasterFifoErrFlag* FIFO error flag.  
*kLPI2C\_MasterPinLowTimeoutFlag* Pin low timeout flag.  
*kLPI2C\_MasterDataMatchFlag* Data match flag.  
*kLPI2C\_MasterBusyFlag* Master busy flag.  
*kLPI2C\_MasterBusBusyFlag* Bus busy flag.  
*kLPI2C\_MasterClearFlags* All flags which are cleared by the driver upon starting a transfer.  
*kLPI2C\_MasterIrqFlags* IRQ sources enabled by the non-blocking transactional API.  
*kLPI2C\_MasterErrorFlags* Errors to check for.

#### 22.6.4.2 enum lpi2c\_direction\_t

Enumerator

***kLPI2C\_Write*** Master transmit.

***kLPI2C\_Read*** Master receive.

#### 22.6.4.3 enum lpi2c\_master\_pin\_config\_t

Enumerator

***kLPI2C\_2PinOpenDrain*** LPI2C Configured for 2-pin open drain mode.

***kLPI2C\_2PinOutputOnly*** LPI2C Configured for 2-pin output only mode (ultra-fast mode)

***kLPI2C\_2PinPushPull*** LPI2C Configured for 2-pin push-pull mode.

***kLPI2C\_4PinPushPull*** LPI2C Configured for 4-pin push-pull mode.

***kLPI2C\_2PinOpenDrainWithSeparateSlave*** LPI2C Configured for 2-pin open drain mode with separate LPI2C slave.

***kLPI2C\_2PinOutputOnlyWithSeparateSlave*** LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave.

***kLPI2C\_2PinPushPullWithSeparateSlave*** LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave.

***kLPI2C\_4PinPushPullWithInvertedOutput*** LPI2C Configured for 4-pin push-pull mode(inverted outputs)

#### 22.6.4.4 enum lpi2c\_host\_request\_source\_t

Enumerator

***kLPI2C\_HostRequestExternalPin*** Select the LPI2C\_HREQ pin as the host request input.

***kLPI2C\_HostRequestInputTrigger*** Select the input trigger as the host request input.

#### 22.6.4.5 enum lpi2c\_host\_request\_polarity\_t

Enumerator

***kLPI2C\_HostRequestPinActiveLow*** Configure the LPI2C\_HREQ pin active low.

***kLPI2C\_HostRequestPinActiveHigh*** Configure the LPI2C\_HREQ pin active high.

#### 22.6.4.6 enum lpi2c\_data\_match\_config\_mode\_t

Enumerator

***kLPI2C\_MatchDisabled*** LPI2C Match Disabled.

***kLPI2C\_1stWordEqualsM0OrM1*** LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1.

***kLPI2C\_AnyWordEqualsM0OrM1*** LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1.

***kLPI2C\_1stWordEqualsM0And2ndWordEqualsM1*** LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1.

***kLPI2C\_AnyWordEqualsM0AndNextWordEqualsM1*** LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1.

***kLPI2C\_1stWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1.

***kLPI2C\_AnyWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1.

#### 22.6.4.7 enum \_lpi2c\_master\_transfer\_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the [`\_lpi2c\_master\_transfer::flags`](#) field.

Enumerator

***kLPI2C\_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.

***kLPI2C\_TransferNoStartFlag*** Don't send a start condition, address, and sub address.

***kLPI2C\_TransferRepeatedStartFlag*** Send a repeated start condition.

***kLPI2C\_TransferNoStopFlag*** Don't send a stop condition.

#### 22.6.5 Function Documentation

##### 22.6.5.1 void LPI2C\_MasterGetDefaultConfig ( `lpi2c_master_config_t * masterConfig` )

This function provides the following default configuration for the LPI2C master peripheral:

```
* masterConfig->enableMaster = true;
* masterConfig->debugEnable = false;
* masterConfig->ignoreAck = false;
* masterConfig->pinConfig = kLPI2C_2PinOpenDrain;
* masterConfig->baudRate_Hz = 1000000U;
* masterConfig->busIdleTimeout_ns = 0;
* masterConfig->pinLowTimeout_ns = 0;
* masterConfig->sdaGlitchFilterWidth_ns = 0;
* masterConfig->sclGlitchFilterWidth_ns = 0;
* masterConfig->hostRequest.enable = false;
* masterConfig->hostRequest.source = kLPI2C_HostRequestExternalPin;
* masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with [`LPI2C\_MasterInit\(\)`](#).

Parameters

|            |                     |                                                                                                            |
|------------|---------------------|------------------------------------------------------------------------------------------------------------|
| <i>out</i> | <i>masterConfig</i> | User provided configuration structure for default values. Refer to <a href="#">lpi2c_master_config_t</a> . |
|------------|---------------------|------------------------------------------------------------------------------------------------------------|

#### 22.6.5.2 void LPI2C\_MasterInit ( LPI2C\_Type \* *base*, const lpi2c\_master\_config\_t \* *masterConfig*, uint32\_t *sourceClock\_Hz* )

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

|                       |                                                                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.                                                                                                         |
| <i>masterConfig</i>   | User provided peripheral configuration. Use <a href="#">LPI2C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.            |

#### 22.6.5.3 void LPI2C\_MasterDeinit ( LPI2C\_Type \* *base* )

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

#### 22.6.5.4 void LPI2C\_MasterConfigureDataMatch ( LPI2C\_Type \* *base*, const lpi2c\_data\_match\_config\_t \* *matchConfig* )

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | The LPI2C peripheral base address.   |
| <i>matchConfig</i> | Settings for the data match feature. |

#### 22.6.5.5 status\_t LPI2C\_MasterCheckAndClearError ( LPI2C\_Type \* *base*, uint32\_t *status* )

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.               |
| <i>status</i> | Current status flags value that will be checked. |

Return values

|                                      |  |
|--------------------------------------|--|
| <i>kStatus_Success</i>               |  |
| <i>kStatus_LPI2C_PinLow-Timeout</i>  |  |
| <i>kStatus_LPI2C_ArbitrationLost</i> |  |
| <i>kStatus_LPI2C_Nak</i>             |  |
| <i>kStatus_LPI2C_FifoError</i>       |  |

### 22.6.5.6 **status\_t LPI2C\_CheckForBusyBus ( LPI2C\_Type \* *base* )**

A busy bus is allowed if we are the one driving it.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Return values

|                           |  |
|---------------------------|--|
| <i>kStatus_Success</i>    |  |
| <i>kStatus_LPI2C_Busy</i> |  |

### 22.6.5.7 **static void LPI2C\_MasterReset ( LPI2C\_Type \* *base* ) [inline], [static]**

Restores the LPI2C master peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

### 22.6.5.8 **static void LPI2C\_MasterEnable ( LPI2C\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                     |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as master. |

### 22.6.5.9 static uint32\_t LPI2C\_MasterGetStatusFlags ( LPI2C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_lpi2c\\_master\\_flags](#)

### 22.6.5.10 static void LPI2C\_MasterClearStatusFlags ( LPI2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- [kLPI2C\\_MasterEndOfPacketFlag](#)
- [kLPI2C\\_MasterStopDetectFlag](#)
- [kLPI2C\\_MasterNackDetectFlag](#)
- [kLPI2C\\_MasterArbitrationLostFlag](#)
- [kLPI2C\\_MasterFifoErrFlag](#)
- [kLPI2C\\_MasterPinLowTimeoutFlag](#)
- [kLPI2C\\_MasterDataMatchFlag](#)

Attempts to clear other flags has no effect.

Parameters

|                   |                                                                                                                                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                    |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_MasterGetStatusFlags()</a> . |

See Also

[\\_lpi2c\\_master\\_flags](#).

#### 22.6.5.11 static void LPI2C\_MasterEnableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 22.6.5.12 static void LPI2C\_MasterDisableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be disabled as interrupts.

Parameters

|                      |                                                                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                     |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_lpi2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 22.6.5.13 static uint32\_t LPI2C\_MasterGetEnabledInterrupts ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

A bitmask composed of \_lpi2c\_master\_flags enumerators OR'd together to indicate the set of enabled interrupts.

#### 22.6.5.14 static void LPI2C\_MasterEnableDMA ( LPI2C\_Type \* *base*, bool *enableTx*, bool *enableRx* ) [inline], [static]

Parameters

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.                                             |
| <i>enableTx</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |
| <i>enableRx</i> | Enable flag for receive DMA request. Pass true for enable, false for disable.  |

#### 22.6.5.15 static uint32\_t LPI2C\_MasterGetTxFifoAddress ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The LPI2C Master Transmit Data Register address.

#### 22.6.5.16 static uint32\_t LPI2C\_MasterGetRxFifoAddress ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The LPI2C Master Receive Data Register address.

#### 22.6.5.17 static void LPI2C\_MasterSetWatermarks ( *LPI2C\_Type* \* *base*, *size\_t* *txWords*, *size\_t* *rxWords* ) [inline], [static]

Parameters

|                |                                                                                                                                                                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                                                                                          |
| <i>txWords</i> | Transmit FIFO watermark value in words. The <a href="#">kLPI2C_MasterTxReadyFlag</a> flag is set whenever the number of words in the transmit FIFO is equal or less than <i>txWords</i> . Writing a value equal or greater than the FIFO size is truncated. |
| <i>rxWords</i> | Receive FIFO watermark value in words. The <a href="#">kLPI2C_MasterRxReadyFlag</a> flag is set whenever the number of words in the receive FIFO is greater than <i>rxWords</i> . Writing a value equal or greater than the FIFO size is truncated.         |

#### 22.6.5.18 static void LPI2C\_MasterGetFifoCounts ( *LPI2C\_Type* \* *base*, *size\_t* \* *rxCount*, *size\_t* \* *txCount* ) [inline], [static]

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | The LPI2C peripheral base address.                                                                                           |
| out | <i>txCount</i> | Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.  |

#### 22.6.5.19 void LPI2C\_MasterSetBaudRate ( *LPI2C\_Type* \* *base*, *uint32\_t* *sourceClock\_Hz*, *uint32\_t* *baudRate\_Hz* )

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

## Note

Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.         |
| <i>sourceClock_Hz</i> | LPI2C functional clock frequency in Hertz. |
| <i>baudRate_Hz</i>    | Requested bus frequency in Hertz.          |

#### 22.6.5.20 static bool LPI2C\_MasterGetBusIdleState ( LPI2C\_Type \* *base* ) [inline], [static]

Requires the master mode to be enabled.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

#### 22.6.5.21 status\_t LPI2C\_MasterStart ( LPI2C\_Type \* *base*, uint8\_t *address*, lpi2c\_direction\_t *dir* )

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

|                |                                                                                                                                                                                     |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                  |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                          |
| <i>dir</i>     | Master transfer direction, either <a href="#">kLPI2C_Read</a> or <a href="#">kLPI2C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                           |                                                                           |
|---------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | START signal and address were successfully enqueued in the transmit FIFO. |
| <i>kStatus_LPI2C_Busy</i> | Another master is currently utilizing the bus.                            |

### 22.6.5.22 static status\_t LPI2C\_MasterRepeatedStart ( LPI2C\_Type \* *base*, uint8\_t *address*, lpi2c\_direction\_t *dir* ) [inline], [static]

This function is used to send a Repeated START signal when a transfer is already in progress. Like [LPI2C\\_MasterStart\(\)](#), it also sends the specified 7-bit address.

Note

This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

|                |                                                                                                                                                                                     |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                  |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                          |
| <i>dir</i>     | Master transfer direction, either <a href="#">kLPI2C_Read</a> or <a href="#">kLPI2C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                           |                                                                                    |
|---------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | Repeated START signal and address were successfully enqueued in the transmit FIFO. |
| <i>kStatus_LPI2C_Busy</i> | Another master is currently utilizing the bus.                                     |

### 22.6.5.23 status\_t LPI2C\_MasterSend ( LPI2C\_Type \* *base*, void \* *txBuff*, size\_t *txSize* )

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns [kStatus\\_LPI2C\\_Nak](#).

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                 |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>               | Data was sent successfully.                        |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or over run.                        |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

#### 22.6.5.24 status\_t LPI2C\_MasterReceive ( LPI2C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize* )

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                 |
| <i>rxBuff</i> | The pointer to the data to be transferred.         |
| <i>rxSize</i> | The length in bytes of the data to be transferred. |

Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>               | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                         |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

#### 22.6.5.25 status\_t LPI2C\_MasterStop ( LPI2C\_Type \* *base* )

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Return values

|                                      |                                                                                  |
|--------------------------------------|----------------------------------------------------------------------------------|
| <i>kStatus_Success</i>               | The STOP signal was successfully sent on the bus and the transaction terminated. |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.                                   |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte.                               |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                                                       |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                                                          |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.                                |

#### 22.6.5.26 `status_t LPI2C_MasterTransferBlocking ( LPI2C_Type * base, Ipi2c_master_transfer_t * transfer )`

Note

The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address. |
| <i>transfer</i> | Pointer to the transfer structure. |

Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>               | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                         |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

**22.6.5.27 void LPI2C\_MasterTransferCreateHandle ( *LPI2C\_Type* \* *base*,  
*lpi2c\_master\_handle\_t* \* *handle*, *lpi2c\_master\_transfer\_callback\_t* *callback*,  
*void* \* *userData* )**

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_MasterTransferAbort\(\)](#) API shall be called.

#### Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

#### Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C master driver handle.                   |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

**22.6.5.28 status\_t LPI2C\_MasterTransferNonBlocking ( *LPI2C\_Type* \* *base*,  
*lpi2c\_master\_handle\_t* \* *handle*, *lpi2c\_master\_transfer\_t* \* *transfer* )**

#### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

#### Return values

|                           |                                                                                                             |
|---------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | The transaction was started successfully.                                                                   |
| <i>kStatus_LPI2C_Busy</i> | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |

**22.6.5.29 status\_t LPI2C\_MasterTransferGetCount ( *LPI2C\_Type* \* *base*,  
*lpi2c\_master\_handle\_t* \* *handle*, *size\_t* \* *count* )**

Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                  |
|     | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

Return values

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

### 22.6.5.30 void LPI2C\_MasterTransferAbort ( LPI2C\_Type \* *base*, Ipi2c\_master\_handle\_t \* *handle* )

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

### 22.6.5.31 void LPI2C\_MasterTransferHandleIRQ ( LPI2C\_Type \* *base*, void \* *Ipi2cMasterHandle* )

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

|                           |                                            |
|---------------------------|--------------------------------------------|
| <i>base</i>               | The LPI2C peripheral base address.         |
| <i>Ipi2cMaster-Handle</i> | Pointer to the LPI2C master driver handle. |

## 22.7 LPI2C Slave Driver

### 22.7.1 Overview

#### Data Structures

- struct `lpi2c_slave_config_t`  
*Structure with settings to initialize the LPI2C slave module. [More...](#)*
- struct `lpi2c_slave_transfer_t`  
*LPI2C slave transfer structure. [More...](#)*
- struct `lpi2c_slave_handle_t`  
*LPI2C slave handle structure. [More...](#)*

#### Typedefs

- typedef void(\* `lpi2c_slave_transfer_callback_t`)`(LPI2C_Type *base, lpi2c_slave_transfer_t *transfer, void *userData)`  
*Slave event callback function pointer type.*

#### Enumerations

- enum `_lpi2c_slave_flags` {
   
`kLPI2C_SlaveTxReadyFlag` = LPI2C\_SSR\_TDF\_MASK,  
`kLPI2C_SlaveRxReadyFlag` = LPI2C\_SSR\_RDF\_MASK,  
`kLPI2C_SlaveAddressValidFlag` = LPI2C\_SSR\_AVF\_MASK,  
`kLPI2C_SlaveTransmitAckFlag` = LPI2C\_SSR\_TAF\_MASK,  
`kLPI2C_SlaveRepeatedStartDetectFlag` = LPI2C\_SSR\_RSF\_MASK,  
`kLPI2C_SlaveStopDetectFlag` = LPI2C\_SSR\_SDF\_MASK,  
`kLPI2C_SlaveBitErrFlag` = LPI2C\_SSR\_BEF\_MASK,  
`kLPI2C_SlaveFifoErrFlag` = LPI2C\_SSR\_FEF\_MASK,  
`kLPI2C_SlaveAddressMatch0Flag` = LPI2C\_SSR\_AM0F\_MASK,  
`kLPI2C_SlaveAddressMatch1Flag` = LPI2C\_SSR\_AM1F\_MASK,  
`kLPI2C_SlaveGeneralCallFlag` = LPI2C\_SSR\_GCF\_MASK,  
`kLPI2C_SlaveBusyFlag` = LPI2C\_SSR\_SBF\_MASK,  
`kLPI2C_SlaveBusBusyFlag` = LPI2C\_SSR\_BBF\_MASK,  
`kLPI2C_SlaveClearFlags`,  
`kLPI2C_SlaveIrqFlags`,  
`kLPI2C_SlaveErrorFlags` = `kLPI2C_SlaveFifoErrFlag | kLPI2C_SlaveBitErrFlag` }
   
*LPI2C slave peripheral flags.*
- enum `lpi2c_slave_address_match_t` {
   
`kLPI2C_MatchAddress0` = 0U,  
`kLPI2C_MatchAddress0OrAddress1` = 2U,  
`kLPI2C_MatchAddress0ThroughAddress1` = 6U }
   
*LPI2C slave address match options.*

- enum `lpi2c_slave_transfer_event_t` {
   
  `kLPI2C_SlaveAddressMatchEvent` = 0x01U,  
`kLPI2C_SlaveTransmitEvent` = 0x02U,  
`kLPI2C_SlaveReceiveEvent` = 0x04U,  
`kLPI2C_SlaveTransmitAckEvent` = 0x08U,  
`kLPI2C_SlaveRepeatedStartEvent` = 0x10U,  
`kLPI2C_SlaveCompletionEvent` = 0x20U,  
`kLPI2C_SlaveAllEvents` }

*Set of events sent to the callback for non blocking slave transfers.*

## Slave initialization and deinitialization

- void `LPI2C_SlaveGetDefaultConfig` (`lpi2c_slave_config_t` \*slaveConfig)  
*Provides a default configuration for the LPI2C slave peripheral.*
- void `LPI2C_SlaveInit` (`LPI2C_Type` \*base, const `lpi2c_slave_config_t` \*slaveConfig, `uint32_t` sourceClock\_Hz)  
*Initializes the LPI2C slave peripheral.*
- void `LPI2C_SlaveDeinit` (`LPI2C_Type` \*base)  
*Deinitializes the LPI2C slave peripheral.*
- static void `LPI2C_SlaveReset` (`LPI2C_Type` \*base)  
*Performs a software reset of the LPI2C slave peripheral.*
- static void `LPI2C_SlaveEnable` (`LPI2C_Type` \*base, bool enable)  
*Enables or disables the LPI2C module as slave.*

## Slave status

- static `uint32_t` `LPI2C_SlaveGetStatusFlags` (`LPI2C_Type` \*base)  
*Gets the LPI2C slave status flags.*
- static void `LPI2C_SlaveClearStatusFlags` (`LPI2C_Type` \*base, `uint32_t` statusMask)  
*Clears the LPI2C status flag state.*

## Slave interrupts

- static void `LPI2C_SlaveEnableInterrupts` (`LPI2C_Type` \*base, `uint32_t` interruptMask)  
*Enables the LPI2C slave interrupt requests.*
- static void `LPI2C_SlaveDisableInterrupts` (`LPI2C_Type` \*base, `uint32_t` interruptMask)  
*Disables the LPI2C slave interrupt requests.*
- static `uint32_t` `LPI2C_SlaveGetEnabledInterrupts` (`LPI2C_Type` \*base)  
*Returns the set of currently enabled LPI2C slave interrupt requests.*

## Slave DMA control

- static void `LPI2C_SlaveEnableDMA` (`LPI2C_Type` \*base, bool enableAddressValid, bool enableRx, bool enableTx)

*Enables or disables the LPI2C slave peripheral DMA requests.*

## Slave bus operations

- static bool [LPI2C\\_SlaveGetBusIdleState](#) (LPI2C\_Type \*base)  
*Returns whether the bus is idle.*
- static void [LPI2C\\_SlaveTransmitAck](#) (LPI2C\_Type \*base, bool ackOrNack)  
*Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.*
- static void [LPI2C\\_SlaveEnableAckStall](#) (LPI2C\_Type \*base, bool enable)  
*Enables or disables ACKSTALL.*
- static uint32\_t [LPI2C\\_SlaveGetReceivedAddress](#) (LPI2C\_Type \*base)  
*Returns the slave address sent by the I2C master.*
- status\_t [LPI2C\\_SlaveSend](#) (LPI2C\_Type \*base, void \*txBuff, size\_t txSize, size\_t \*actualTxSize)  
*Performs a polling send transfer on the I2C bus.*
- status\_t [LPI2C\\_SlaveReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize, size\_t \*actualRxSize)  
*Performs a polling receive transfer on the I2C bus.*

## Slave non-blocking

- void [LPI2C\\_SlaveTransferCreateHandle](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, [lpi2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the LPI2C slave non-blocking APIs.*
- status\_t [LPI2C\\_SlaveTransferNonBlocking](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- status\_t [LPI2C\\_SlaveTransferGetCount](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer status during a non-blocking transfer.*
- void [LPI2C\\_SlaveTransferAbort](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Aborts the slave non-blocking transfers.*

## Slave IRQ handler

- void [LPI2C\\_SlaveTransferHandleIRQ](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Reusable routine to handle slave interrupts.*

### 22.7.2 Data Structure Documentation

#### 22.7.2.1 struct lpi2c\_slave\_config\_t

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_SlaveGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

## Data Fields

- bool `enableSlave`  
*Enable slave mode.*
- uint8\_t `address0`  
*Slave's 7-bit address.*
- uint8\_t `address1`  
*Alternate slave 7-bit address.*
- `lpi2c_slave_address_match_t addressMatchMode`  
*Address matching options.*
- bool `filterDozeEnable`  
*Enable digital glitch filter in doze mode.*
- bool `filterEnable`  
*Enable digital glitch filter.*
- bool `enableGeneralCall`  
*Enable general call address matching.*
- struct {
  - bool `enableAck`  
*Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data bytes.*
  - bool `enableTx`  
*Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.*
  - bool `enableRx`  
*Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.*
  - bool `enableAddress`  
*Enables SCL clock stretching when the address valid flag is asserted.*
} `sclStall`  
*SCL stall enable options.*
- bool `ignoreAck`  
*Continue transfers after a NACK is detected.*
- bool `enableReceivedAddressRead`  
*Enable reading the address received address as the first byte of data.*
- uint32\_t `sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SDA signal.*
- uint32\_t `sclGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SCL signal.*
- uint32\_t `dataValidDelay_ns`  
*Width in nanoseconds of the data valid delay.*
- uint32\_t `clockHoldTime_ns`  
*Width in nanoseconds of the clock hold time.*

## Field Documentation

- (1) `bool lpi2c_slave_config_t::enableSlave`
- (2) `uint8_t lpi2c_slave_config_t::address0`
- (3) `uint8_t lpi2c_slave_config_t::address1`
- (4) `lpi2c_slave_address_match_t lpi2c_slave_config_t::addressMatchMode`
- (5) `bool lpi2c_slave_config_t::filterDozeEnable`
- (6) `bool lpi2c_slave_config_t::filterEnable`
- (7) `bool lpi2c_slave_config_t::enableGeneralCall`
- (8) `bool lpi2c_slave_config_t::enableAck`

Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

- (9) `bool lpi2c_slave_config_t::enableTx`
- (10) `bool lpi2c_slave_config_t::enableRx`
- (11) `bool lpi2c_slave_config_t::enableAddress`
- (12) `struct { ... } lpi2c_slave_config_t::sclStall`
- (13) `bool lpi2c_slave_config_t::ignoreAck`
- (14) `bool lpi2c_slave_config_t::enableReceivedAddressRead`
- (15) `uint32_t lpi2c_slave_config_t::sdaGlitchFilterWidth_ns`

Set to 0 to disable.

- (16) `uint32_t lpi2c_slave_config_t::sclGlitchFilterWidth_ns`

Set to 0 to disable.

- (17) `uint32_t lpi2c_slave_config_t::dataValidDelay_ns`
- (18) `uint32_t lpi2c_slave_config_t::clockHoldTime_ns`

### 22.7.2.2 struct lpi2c\_slave\_transfer\_t

#### Data Fields

- `lpi2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*

- `uint8_t receivedAddress`  
*Matching address send by master.*
- `uint8_t * data`  
*Transfer buffer.*
- `size_t dataSize`  
*Transfer size.*
- `status_t completionStatus`  
*Success or error code describing how the transfer completed.*
- `size_t transferredCount`  
*Number of bytes actually transferred since start or last repeated start.*

## Field Documentation

(1) `lpi2c_slave_transfer_event_t lpi2c_slave_transfer_t::event`

(2) `uint8_t lpi2c_slave_transfer_t::receivedAddress`

(3) `status_t lpi2c_slave_transfer_t::completionStatus`

Only applies for [kLPI2C\\_SlaveCompletionEvent](#).

(4) `size_t lpi2c_slave_transfer_t::transferredCount`

### 22.7.2.3 struct \_lpi2c\_slave\_handle

#### Note

The contents of this structure are private and subject to change.

## Data Fields

- `lpi2c_slave_transfer_t transfer`  
*LPI2C slave transfer copy.*
- `bool isBusy`  
*Whether transfer is busy.*
- `bool wasTransmit`  
*Whether the last transfer was a transmit.*
- `uint32_t eventMask`  
*Mask of enabled events.*
- `uint32_t transferredCount`  
*Count of bytes transferred.*
- `lpi2c_slave_transfer_callback_t callback`  
*Callback function called at transfer event.*
- `void * userData`  
*Callback parameter passed to callback.*

## Field Documentation

- (1) `lpi2c_slave_transfer_t lpi2c_slave_handle_t::transfer`
- (2) `bool lpi2c_slave_handle_t::isBusy`
- (3) `bool lpi2c_slave_handle_t::wasTransmit`
- (4) `uint32_t lpi2c_slave_handle_t::eventMask`
- (5) `uint32_t lpi2c_slave_handle_t::transferredCount`
- (6) `lpi2c_slave_transfer_callback_t lpi2c_slave_handle_t::callback`
- (7) `void* lpi2c_slave_handle_t::userData`

## 22.7.3 Typedef Documentation

### 22.7.3.1 `typedef void(* lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, lpi2c_slave_transfer_t *transfer, void *userData)`

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C\_SlaveSetCallback() function after you have created a handle.

## Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for the LPI2C instance on which the event occurred.                     |
| <i>transfer</i> | Pointer to transfer descriptor containing values passed to and/or from the callback. |
| <i>userData</i> | Arbitrary pointer-sized value passed from the application.                           |

**22.7.4 Enumeration Type Documentation****22.7.4.1 enum \_lpi2c\_slave\_flags**

The following status register flags can be cleared:

- [kLPI2C\\_SlaveRepeatedStartDetectFlag](#)
- [kLPI2C\\_SlaveStopDetectFlag](#)
- [kLPI2C\\_SlaveBitErrFlag](#)
- [kLPI2C\\_SlaveFifoErrFlag](#)

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

## Note

These enumerations are meant to be OR'd together to form a bit mask.

## Enumerator

- kLPI2C\_SlaveTxReadyFlag* Transmit data flag.  
*kLPI2C\_SlaveRxReadyFlag* Receive data flag.  
*kLPI2C\_SlaveAddressValidFlag* Address valid flag.  
*kLPI2C\_SlaveTransmitAckFlag* Transmit ACK flag.  
*kLPI2C\_SlaveRepeatedStartDetectFlag* Repeated start detect flag.  
*kLPI2C\_SlaveStopDetectFlag* Stop detect flag.  
*kLPI2C\_SlaveBitErrFlag* Bit error flag.  
*kLPI2C\_SlaveFifoErrFlag* FIFO error flag.  
*kLPI2C\_SlaveAddressMatch0Flag* Address match 0 flag.  
*kLPI2C\_SlaveAddressMatch1Flag* Address match 1 flag.  
*kLPI2C\_SlaveGeneralCallFlag* General call flag.  
*kLPI2C\_SlaveBusyFlag* Master busy flag.  
*kLPI2C\_SlaveBusBusyFlag* Bus busy flag.  
*kLPI2C\_SlaveClearFlags* All flags which are cleared by the driver upon starting a transfer.  
*kLPI2C\_SlaveIrqFlags* IRQ sources enabled by the non-blocking transactional API.  
*kLPI2C\_SlaveErrorFlags* Errors to check for.

### 22.7.4.2 enum lpi2c\_slave\_address\_match\_t

Enumerator

*kLPI2C\_MatchAddress0* Match only address 0.

*kLPI2C\_MatchAddress0OrAddress1* Match either address 0 or address 1.

*kLPI2C\_MatchAddress0ThroughAddress1* Match a range of slave addresses from address 0 through address 1.

### 22.7.4.3 enum lpi2c\_slave\_transfer\_event\_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to [LPI2C\\_SlaveTransferNonBlocking\(\)](#) in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

*kLPI2C\_SlaveAddressMatchEvent* Received the slave address after a start or repeated start.

*kLPI2C\_SlaveTransmitEvent* Callback is requested to provide data to transmit (slave-transmitter role).

*kLPI2C\_SlaveReceiveEvent* Callback is requested to provide a buffer in which to place received data (slave-receiver role).

*kLPI2C\_SlaveTransmitAckEvent* Callback needs to either transmit an ACK or NACK.

*kLPI2C\_SlaveRepeatedStartEvent* A repeated start was detected.

*kLPI2C\_SlaveCompletionEvent* A stop was detected, completing the transfer.

*kLPI2C\_SlaveAllEvents* Bit mask of all available events.

## 22.7.5 Function Documentation

### 22.7.5.1 void LPI2C\_SlaveGetDefaultConfig ( lpi2c\_slave\_config\_t \* *slaveConfig* )

This function provides the following default configuration for the LPI2C slave peripheral:

```
* slaveConfig->enableSlave = true;
* slaveConfig->address0 = 0U;
* slaveConfig->address1 = 0U;
* slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
* slaveConfig->filterDozeEnable = true;
* slaveConfig->filterEnable = true;
* slaveConfig->enableGeneralCall = false;
* slaveConfig->sclStall.enableAck = false;
* slaveConfig->sclStall.enableTx = true;
* slaveConfig->sclStall.enableRx = true;
* slaveConfig->sclStall.enableAddress = true;
```

```

* slaveConfig->ignoreAck = false;
* slaveConfig->enableReceivedAddressRead = false;
* slaveConfig->sdaGlitchFilterWidth_ns = 0;
* slaveConfig->sclGlitchFilterWidth_ns = 0;
* slaveConfig->dataValidDelay_ns = 0;
* slaveConfig->clockHoldTime_ns = 0;
*

```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with [LPI2C\\_SlaveInit\(\)](#). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

|            |                    |                                                                                                                      |
|------------|--------------------|----------------------------------------------------------------------------------------------------------------------|
| <i>out</i> | <i>slaveConfig</i> | User provided configuration structure that is set to default values. Refer to <a href="#">lpi2c_slave_config_t</a> . |
|------------|--------------------|----------------------------------------------------------------------------------------------------------------------|

#### 22.7.5.2 void LPI2C\_SlaveInit ( LPI2C\_Type \* *base*, const lpi2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *sourceClock\_Hz* )

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

|                       |                                                                                                                                           |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.                                                                                                        |
| <i>slaveConfig</i>    | User provided peripheral configuration. Use <a href="#">LPI2C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.             |

#### 22.7.5.3 void LPI2C\_SlaveDeinit ( LPI2C\_Type \* *base* )

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

#### 22.7.5.4 static void LPI2C\_SlaveReset ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

#### 22.7.5.5 static void LPI2C\_SlaveEnable ( LPI2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                    |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as slave. |

#### 22.7.5.6 static uint32\_t LPI2C\_SlaveGetStatusFlags ( LPI2C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_lpi2c\\_slave\\_flags](#)

#### 22.7.5.7 static void LPI2C\_SlaveClearStatusFlags ( LPI2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- [kLPI2C\\_SlaveRepeatedStartDetectFlag](#)
- [kLPI2C\\_SlaveStopDetectFlag](#)
- [kLPI2C\\_SlaveBitErrFlag](#)
- [kLPI2C\\_SlaveFifoErrFlag](#)

Attempts to clear other flags has no effect.

Parameters

|                   |                                                                                                                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                                  |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_lpi2c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_SlaveGetStatusFlags()</a> . |

See Also

[\\_lpi2c\\_slave\\_flags](#).

#### 22.7.5.8 static void LPI2C\_SlaveEnableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                   |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 22.7.5.9 static void LPI2C\_SlaveDisableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be disabled as interrupts.

Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 22.7.5.10 static uint32\_t LPI2C\_SlaveGetEnabledInterrupts ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

A bitmask composed of [\\_lpi2c\\_slave\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

#### 22.7.5.11 static void LPI2C\_SlaveEnableDMA ( LPI2C\_Type \* *base*, bool *enableAddressValid*, bool *enableRx*, bool *enableTx* ) [inline], [static]

Parameters

|                           |                                                                                                                                                                    |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | The LPI2C peripheral base address.                                                                                                                                 |
| <i>enableAddressValid</i> | Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request. |
| <i>enableRx</i>           | Enable flag for the receive data DMA request. Pass true for enable, false for disable.                                                                             |
| <i>enableTx</i>           | Enable flag for the transmit data DMA request. Pass true for enable, false for disable.                                                                            |

#### 22.7.5.12 static bool LPI2C\_SlaveGetBusIdleState ( LPI2C\_Type \* *base* ) [inline], [static]

Requires the slave mode to be enabled.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

#### 22.7.5.13 static void LPI2C\_SlaveTransmitAck ( LPI2C\_Type \* *base*, bool *ackOrNack* ) [inline], [static]

Use this function to send an ACK or NAK when the [KLPI2C\\_SlaveTransmitAckFlag](#) is asserted. This only happens if you enable the sclStall.enableAck field of the [lpi2c\\_slave\\_config\\_t](#) configuration structure used to initialize the slave peripheral.

Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | The LPI2C peripheral base address.       |
| <i>ackOrNack</i> | Pass true for an ACK or false for a NAK. |

#### 22.7.5.14 static void LPI2C\_SlaveEnableAckStall ( **LPI2C\_Type** \* *base*, **bool enable** ) [**inline**], [**static**]

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                      |
| <i>enable</i> | True will enable ACKSTALL, false will disable ACKSTALL. |

#### 22.7.5.15 static uint32\_t LPI2C\_SlaveGetReceivedAddress ( **LPI2C\_Type** \* *base* ) [**inline**], [**static**]

This function should only be called if the [kLPI2C\\_SlaveAddressValidFlag](#) is asserted.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

#### 22.7.5.16 status\_t LPI2C\_SlaveSend ( **LPI2C\_Type** \* *base*, **void** \* *txBuff*, **size\_t** *txSize*, **size\_t** \* *actualTxSize* )

Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C peripheral base address.                 |
|     | <i>txBuff</i>       | The pointer to the data to be transferred.         |
|     | <i>txSize</i>       | The length in bytes of the data to be transferred. |
| out | <i>actualTxSize</i> |                                                    |

Returns

Error or success status returned by API.

#### 22.7.5.17 **status\_t LPI2C\_SlaveReceive ( LPI2C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize*, size\_t \* *actualRxSize* )**

Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C peripheral base address.                 |
|     | <i>rxBuff</i>       | The pointer to the data to be transferred.         |
|     | <i>rxSize</i>       | The length in bytes of the data to be transferred. |
| out | <i>actualRxSize</i> |                                                    |

Returns

Error or success status returned by API.

#### 22.7.5.18 **void LPI2C\_SlaveTransferCreateHandle ( LPI2C\_Type \* *base*, Ipi2c\_slave\_handle\_t \* *handle*, Ipi2c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_SlaveTransferAbort\(\)](#) API shall be called.

Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C slave driver handle.                    |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

#### 22.7.5.19 status\_t LPI2C\_SlaveTransferNonBlocking ( **LPI2C\_Type** \* *base*, **lpi2c\_slave\_handle\_t** \* *handle*, **uint32\_t** *eventMask* )

Call this API after calling I2C\_SlaveInit() and [LPI2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to [LPI2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [lpi2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [kLPI2C\\_SlaveTransmitEvent](#) and [kLPI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kLPI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

Parameters

|                  |                                                                                                                                                                                                                                                                                                        |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The LPI2C peripheral base address.                                                                                                                                                                                                                                                                     |
| <i>handle</i>    | Pointer to <a href="#">lpi2c_slave_handle_t</a> structure which stores the transfer state.                                                                                                                                                                                                             |
| <i>eventMask</i> | Bit mask formed by OR'ing together <a href="#">lpi2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kLPI2C_SlaveAllEvents</a> to enable all events. |

Return values

|                                    |                                                           |
|------------------------------------|-----------------------------------------------------------|
| <a href="#">kStatus_Success</a>    | Slave transfers were successfully started.                |
| <a href="#">kStatus_LPI2C_Busy</a> | Slave transfers have already been started on this handle. |

#### 22.7.5.20 status\_t LPI2C\_SlaveTransferGetCount ( **LPI2C\_Type** \* *base*, **lpi2c\_slave\_handle\_t** \* *handle*, **size\_t** \* *count* )

## Parameters

|     |               |                                                                                                       |
|-----|---------------|-------------------------------------------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                                                    |
|     | <i>handle</i> | Pointer to i2c_slave_handle_t structure.                                                              |
| out | <i>count</i>  | Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required. |

## Return values

|                                      |  |
|--------------------------------------|--|
| <i>kStatus_Success</i>               |  |
| <i>kStatus_NoTransferIn-Progress</i> |  |

**22.7.5.21 void LPI2C\_SlaveTransferAbort ( LPI2C\_Type \* *base*, Ipi2c\_slave\_handle\_t \* *handle* )**

## Note

This API could be called at any time to stop slave for handling the bus events.

## Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                         |
| <i>handle</i> | Pointer to lpi2c_slave_handle_t structure which stores the transfer state. |

**22.7.5.22 void LPI2C\_SlaveTransferHandleIRQ ( LPI2C\_Type \* *base*, Ipi2c\_slave\_handle\_t \* *handle* )**

## Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

## Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                         |
| <i>handle</i> | Pointer to lpi2c_slave_handle_t structure which stores the transfer state. |

## 22.8 LPI2C Master DMA Driver

### 22.8.1 Overview

#### Data Structures

- struct `lpi2c_master_edma_handle_t`  
*Driver handle for master DMA APIs. [More...](#)*

#### Typedefs

- `typedef void(* lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, lpi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)`  
*Master DMA completion callback function pointer type.*

#### Master DMA

- `void LPI2C_MasterCreateEDMAHandle (LPI2C_Type *base, lpi2c_master_edma_handle_t *handle, edma_handle_t *rxDmaHandle, edma_handle_t *txDmaHandle, lpi2c_master_edma_transfer_callback_t callback, void *userData)`  
*Create a new handle for the LPI2C master DMA APIs.*
- `status_t LPI2C_MasterTransferEDMA (LPI2C_Type *base, lpi2c_master_edma_handle_t *handle, lpi2c_master_transfer_t *transfer)`  
*Performs a non-blocking DMA-based transaction on the I2C bus.*
- `status_t LPI2C_MasterTransferGetCountEDMA (LPI2C_Type *base, lpi2c_master_edma_handle_t *handle, size_t *count)`  
*Returns number of bytes transferred so far.*
- `status_t LPI2C_MasterTransferAbortEDMA (LPI2C_Type *base, lpi2c_master_edma_handle_t *handle)`  
*Terminates a non-blocking LPI2C master transmission early.*

### 22.8.2 Data Structure Documentation

#### 22.8.2.1 struct \_lpi2c\_master\_edma\_handle

LPI2C master EDMA handle of the transfer.

Note

The contents of this structure are private and subject to change.

#### Data Fields

- `LPI2C_Type * base`

- *LPI2C base pointer.*
- `bool isBusy`  
*Transfer state machine current state.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint16_t commandBuffer [10]`  
*LPI2C command sequence.*
- `lpi2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `lpi2c_master_edma_transfer_callback_t completionCallback`  
*Callback function pointer.*
- `void *userData`  
*Application data passed to callback.*
- `edma_handle_t *rx`  
*Handle for receive DMA channel.*
- `edma_handle_t *tx`  
*Handle for transmit DMA channel.*
- `edma_tcd_t tcds [3]`  
*Software TCD.*

## Field Documentation

- (1) `LPI2C_Type* lpi2c_master_edma_handle_t::base`
- (2) `bool lpi2c_master_edma_handle_t::isBusy`
- (3) `uint8_t lpi2c_master_edma_handle_t::nbytes`
- (4) `uint16_t lpi2c_master_edma_handle_t::commandBuffer[10]`

When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

- (5) `lpi2c_master_transfer_t lpi2c_master_edma_handle_t::transfer`
- (6) `lpi2c_master_edma_transfer_callback_t lpi2c_master_edma_handle_t::completionCallback`
- (7) `void* lpi2c_master_edma_handle_t::userData`
- (8) `edma_handle_t* lpi2c_master_edma_handle_t::rx`
- (9) `edma_handle_t* lpi2c_master_edma_handle_t::tx`
- (10) `edma_tcd_t lpi2c_master_edma_handle_t::tcds[3]`

Three are allocated to provide enough room to align to 32-bytes.

## 22.8.3 Typedef Documentation

**22.8.3.1 `typedef void(* Ipi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, Ipi2c_master_edma_handle_t *handle, status_t completionStatus, void *userData)`**

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [LPI2C\\_MasterCreateEDMAHandle\(\)](#).

Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <i>base</i>              | The LPI2C peripheral base address.                                             |
| <i>handle</i>            | Handle associated with the completed transfer.                                 |
| <i>completion-Status</i> | Either kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                     |

## 22.8.4 Function Documentation

**22.8.4.1 void LPI2C\_MasterCreateEDMAHandle ( LPI2C\_Type \* *base*, Ipi2c\_master\_edma\_handle\_t \* *handle*, edma\_handle\_t \* *rxDmaHandle*, edma\_handle\_t \* *txDmaHandle*, Ipi2c\_master\_edma\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_MasterTransferAbort-EDMA\(\)](#) API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

|     |                    |                                                                                           |
|-----|--------------------|-------------------------------------------------------------------------------------------|
|     | <i>base</i>        | The LPI2C peripheral base address.                                                        |
| out | <i>handle</i>      | Pointer to the LPI2C master driver handle.                                                |
|     | <i>rxDmaHandle</i> | Handle for the eDMA receive channel. Created by the user prior to calling this function.  |
|     | <i>txDmaHandle</i> | Handle for the eDMA transmit channel. Created by the user prior to calling this function. |
|     | <i>callback</i>    | User provided pointer to the asynchronous callback function.                              |
|     | <i>userData</i>    | User provided pointer to the application callback data.                                   |

**22.8.4.2 status\_t LPI2C\_MasterTransferEDMA ( LPI2C\_Type \* *base*, Ipi2c\_master\_edma\_handle\_t \* *handle*, Ipi2c\_master\_transfer\_t \* *transfer* )**

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

Return values

|                           |                                                                                                          |
|---------------------------|----------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>    | The transaction was started successfully.                                                                |
| <i>kStatus_LPI2C_Busy</i> | Either another master is currently utilizing the bus, or another DMA transaction is already in progress. |

#### 22.8.4.3 status\_t LPI2C\_MasterTransferGetCountEDMA ( **LPI2C\_Type \* base,**                   *lpi2c\_master\_edma\_handle\_t \* handle, size\_t \* count* )

Parameters

|            |               |                                                                     |
|------------|---------------|---------------------------------------------------------------------|
|            | <i>base</i>   | The LPI2C peripheral base address.                                  |
|            | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| <i>out</i> | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>              |                                                       |
| <i>kStatus_NoTransferInProgress</i> | There is not a DMA transaction currently in progress. |

#### 22.8.4.4 status\_t LPI2C\_MasterTransferAbortEDMA ( **LPI2C\_Type \* base,**                   *lpi2c\_master\_edma\_handle\_t \* handle* )

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

## Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

## Return values

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>    | A transaction was successfully aborted.               |
| <i>kStatus_LPI2C_Idle</i> | There is not a DMA transaction currently in progress. |

## 22.9 LPI2C FreeRTOS Driver

### 22.9.1 Overview

#### Driver version

- `#define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`  
*LPI2C FreeRTOS driver version.*

#### LPI2C RTOS Operation

- `status_t LPI2C_RRTOS_Init (lpi2c_rtos_handle_t *handle, LPI2C_Type *base, const lpi2c_master_config_t *masterConfig, uint32_t srcClock_Hz)`  
*Initializes LPI2C.*
- `status_t LPI2C_RRTOS_Deinit (lpi2c_rtos_handle_t *handle)`  
*Deinitializes the LPI2C.*
- `status_t LPI2C_RRTOS_Transfer (lpi2c_rtos_handle_t *handle, lpi2c_master_transfer_t *transfer)`  
*Performs I2C transfer.*

### 22.9.2 Macro Definition Documentation

#### 22.9.2.1 `#define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`

### 22.9.3 Function Documentation

#### 22.9.3.1 `status_t LPI2C_RRTOS_Init ( lpi2c_rtos_handle_t * handle, LPI2C_Type * base, const lpi2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the LPI2C module and related RTOS context.

Parameters

|                           |                                                                            |
|---------------------------|----------------------------------------------------------------------------|
| <code>handle</code>       | The RTOS LPI2C handle, the pointer to an allocated space for RTOS context. |
| <code>base</code>         | The pointer base address of the LPI2C instance to initialize.              |
| <code>masterConfig</code> | Configuration structure to set-up LPI2C in master mode.                    |
| <code>srcClock_Hz</code>  | Frequency of input clock of the LPI2C module.                              |

Returns

status of the operation.

### 22.9.3.2 status\_t LPI2C\_RTOS\_Deinit ( *lpi2c\_rtos\_handle\_t \* handle* )

This function deinitializes the LPI2C module and related RTOS context.

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPI2C handle. |
|---------------|------------------------|

### 22.9.3.3 status\_t LPI2C\_RTOs\_Transfer ( *lpi2c\_rtos\_handle\_t \* handle,* *lpi2c\_master\_transfer\_t \* transfer* )

This function performs an I2C transfer using LPI2C module according to data given in the transfer structure.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS LPI2C handle.                        |
| <i>transfer</i> | Structure specifying the transfer parameters. |

Returns

status of the operation.

## 22.10 LPI2C CMSIS Driver

This section describes the programming interface of the LPI2C 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 see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The LPI2C 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.

### 22.10.1 LPI2C CMSIS Driver

#### 22.10.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
 if (event == ARM_I2C_EVENT_TRANSFER_DONE)
 {
 g_MasterCompletionFlag = true;
 }
}
/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
Driver_I2C0.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

#### 22.10.1.2 Master Operation in DMA transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
 /* Transfer done */
 if (event == ARM_I2C_EVENT_TRANSFER_DONE)
 {
 g_MasterCompletionFlag = true;
 }
}

/* DMAMux init and EDMA init. */
DMAMUX_Init(EXAMPLE_LPI2C_DMAMUX_BASEADDR);
```

```

edma_config_t edmaConfig;
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(EXAMPLE_LPI2C_DMA_BASEADDR, &edmaConfig);

/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transfer*/
Driver_I2C0.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

### 22.10.1.3 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 23

## LPIT: Low-Power Interrupt Timer

### 23.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Interrupt Timer (LPIT) of MCUXpresso SDK devices.

### 23.2 Function groups

The LPIT driver supports operating the module as a time counter.

#### 23.2.1 Initialization and deinitialization

The function [LPIT\\_Init\(\)](#) initializes the LPIT with specified configurations. The function [LPIT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the LPIT operation in doze mode and debug mode.

The function [LPIT\\_SetupChannel\(\)](#) configures the operation of each LPIT channel.

The function [LPIT\\_Deinit\(\)](#) disables the LPIT module and disables the module clock.

#### 23.2.2 Timer period Operations

The function [LPITR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function [LPIT\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. User can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

#### 23.2.3 Start and Stop timer operations

The function [LPIT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value set earlier via the [LPIT\\_SetPeriod\(\)](#) function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function [LPIT\\_StopTimer\(\)](#) stops the timer counting.

### 23.2.4 Status

Provides functions to get and clear the LPIT status.

### 23.2.5 Interrupt

Provides functions to enable/disable LPIT interrupts and get current enabled interrupts.

## 23.3 Typical use case

### 23.3.1 LPIT tick example

Updates the LPIT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/lpit

## Data Structures

- struct [lpit\\_chnl\\_params\\_t](#)  
*Structure to configure the channel timer. [More...](#)*
- struct [lpit\\_config\\_t](#)  
*LPIT configuration structure. [More...](#)*

## Functions

- static void [LPIT\\_Reset](#) (LPIT\_Type \*base)  
*Performs a software reset on the LPIT module.*

## Driver version

- enum [lpit\\_chnl\\_t](#) {  
  kLPIT\_Chnl\_0 = 0U,  
  kLPIT\_Chnl\_1,  
  kLPIT\_Chnl\_2,  
  kLPIT\_Chnl\_3 }  
*List of LPIT channels.*
- enum [lpit\\_timer\\_modes\\_t](#) {  
  kLPIT\_PeriodicCounter = 0U,  
  kLPIT\_DualPeriodicCounter,  
  kLPIT\_TriggerAccumulator,  
  kLPIT\_InputCapture }  
*Mode options available for the LPIT timer.*
- enum [lpit\\_trigger\\_select\\_t](#) {

```

kLPIT_Trigger_TimerChn0 = 0U,
kLPIT_Trigger_TimerChn1,
kLPIT_Trigger_TimerChn2,
kLPIT_Trigger_TimerChn3,
kLPIT_Trigger_TimerChn4,
kLPIT_Trigger_TimerChn5,
kLPIT_Trigger_TimerChn6,
kLPIT_Trigger_TimerChn7,
kLPIT_Trigger_TimerChn8,
kLPIT_Trigger_TimerChn9,
kLPIT_Trigger_TimerChn10,
kLPIT_Trigger_TimerChn11,
kLPIT_Trigger_TimerChn12,
kLPIT_Trigger_TimerChn13,
kLPIT_Trigger_TimerChn14,
kLPIT_Trigger_TimerChn15 }

```

*Trigger options available.*

- enum `lpit_trigger_source_t` {

```

kLPIT_TriggerSource_External = 0U,
kLPIT_TriggerSource_Internal }
```

*Trigger source options available.*

- enum `lpit_interrupt_enable_t` {

```

kLPIT_Channel0TimerInterruptEnable = (1U << 0),
kLPIT_Channel1TimerInterruptEnable = (1U << 1),
kLPIT_Channel2TimerInterruptEnable = (1U << 2),
kLPIT_Channel3TimerInterruptEnable = (1U << 3) }
```

*List of LPIT interrupts.*

- enum `lpit_status_flags_t` {

```

kLPIT_Channel0TimerFlag = (1U << 0),
kLPIT_Channel1TimerFlag = (1U << 1),
kLPIT_Channel2TimerFlag = (1U << 2),
kLPIT_Channel3TimerFlag = (1U << 3) }
```

*List of LPIT status flags.*

- #define `FSL_LPIT_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)

*Version 2.1.1.*

## Initialization and deinitialization

- void `LPIT_Init` (`LPIT_Type` \*base, const `lpit_config_t` \*config)

*Ungates the LPIT clock and configures the peripheral for a basic operation.*
- void `LPIT_Deinit` (`LPIT_Type` \*base)

*Disables the module and gates the LPIT clock.*
- void `LPIT_GetDefaultConfig` (`lpit_config_t` \*config)

*Fills in the LPIT configuration structure with default settings.*
- `status_t LPIT_SetupChannel` (`LPIT_Type` \*base, `lpit_chnl_t` channel, const `lpit_chnl_params_t` \*chnlSetup)

*Sets up an LPIT channel based on the user's preference.*

## Interrupt Interface

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

## Status Interface

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

## Read and Write the timer period

- static void [LPIT\\_SetTimerPeriod](#) (LPIT\_Type \*base, lpit\_chnl\_t channel, uint32\_t ticks)  
*Sets the timer period in units of count.*
- static void [LPIT\\_SetTimerValue](#) (LPIT\_Type \*base, lpit\_chnl\_t channel, uint32\_t ticks)  
*Sets the timer period in units of count.*
- static uint32\_t [LPIT\\_GetCurrentTimerCount](#) (LPIT\_Type \*base, lpit\_chnl\_t channel)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [LPIT\\_StartTimer](#) (LPIT\_Type \*base, lpit\_chnl\_t channel)  
*Starts the timer counting.*
- static void [LPIT\\_StopTimer](#) (LPIT\_Type \*base, lpit\_chnl\_t channel)  
*Stops the timer counting.*

## 23.4 Data Structure Documentation

### 23.4.1 struct lpit\_chnl\_params\_t

#### Data Fields

- bool [chainChannel](#)  
*true: Timer chained to previous timer; false: Timer not chained*
- [lpit\\_timer\\_modes\\_t timerMode](#)  
*Timers mode of operation.*
- [lpit\\_trigger\\_select\\_t triggerSelect](#)  
*Trigger selection for the timer.*
- [lpit\\_trigger\\_source\\_t triggerSource](#)  
*Decides if we use external or internal trigger.*
- bool [enableReloadOnTrigger](#)  
*true: Timer reloads when a trigger is detected; false: No effect*
- bool [enableStopOnTimeout](#)  
*true: Timer will stop after timeout; false: does not stop after timeout*

- bool `enableStartOnTrigger`  
*true: Timer starts when a trigger is detected; false: decrement immediately*

**Field Documentation**

- (1) `lpit_timer_modes_t lpit_chnl_params_t::timerMode`
- (2) `lpit_trigger_source_t lpit_chnl_params_t::triggerSource`

**23.4.2 struct lpit\_config\_t**

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the [LPIT\\_GetDefaultConfig\(\)](#) function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

**Data Fields**

- bool `enableRunInDebug`  
*true: Timers run in debug mode; false: Timers stop in debug mode*
- bool `enableRunInDoze`  
*true: Timers run in doze mode; false: Timers stop in doze mode*

**23.5 Enumeration Type Documentation****23.5.1 enum lpit\_chnl\_t**

Note

Actual number of available channels is SoC-dependent

Enumerator

- `kLPIT_Chnl_0`** LPIT channel number 0.
- `kLPIT_Chnl_1`** LPIT channel number 1.
- `kLPIT_Chnl_2`** LPIT channel number 2.
- `kLPIT_Chnl_3`** LPIT channel number 3.

**23.5.2 enum lpit\_timer\_modes\_t**

Enumerator

- `kLPIT_PeriodicCounter`** Use the all 32-bits, counter loads and decrements to zero.
- `kLPIT_DualPeriodicCounter`** Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement.

***kLPIT\_TriggerAccumulator*** Counter loads on first trigger and decrements on each trigger.

***kLPIT\_InputCapture*** Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

### 23.5.3 enum lpit\_trigger\_select\_t

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Enumerator

***kLPIT\_Trigger\_TimerChn0*** Channel 0 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn1*** Channel 1 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn2*** Channel 2 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn3*** Channel 3 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn4*** Channel 4 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn5*** Channel 5 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn6*** Channel 6 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn7*** Channel 7 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn8*** Channel 8 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn9*** Channel 9 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn10*** Channel 10 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn11*** Channel 11 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn12*** Channel 12 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn13*** Channel 13 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn14*** Channel 14 is selected as a trigger source.

***kLPIT\_Trigger\_TimerChn15*** Channel 15 is selected as a trigger source.

### 23.5.4 enum lpit\_trigger\_source\_t

Enumerator

***kLPIT\_TriggerSource\_External*** Use external trigger input.

***kLPIT\_TriggerSource\_Internal*** Use internal trigger.

### 23.5.5 enum lpit\_interrupt\_enable\_t

## Note

Number of timer channels are SoC-specific. See the SoC Reference Manual.

## Enumerator

|                                           |                            |
|-------------------------------------------|----------------------------|
| <i>kLPIT_Channel0TimerInterruptEnable</i> | Channel 0 Timer interrupt. |
| <i>kLPIT_Channel1TimerInterruptEnable</i> | Channel 1 Timer interrupt. |
| <i>kLPIT_Channel2TimerInterruptEnable</i> | Channel 2 Timer interrupt. |
| <i>kLPIT_Channel3TimerInterruptEnable</i> | Channel 3 Timer interrupt. |

**23.5.6 enum lpit\_status\_flags\_t**

## Note

Number of timer channels are SoC-specific. See the SoC Reference Manual.

## Enumerator

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kLPIT_Channel0TimerFlag</i> | Channel 0 Timer interrupt flag. |
| <i>kLPIT_Channel1TimerFlag</i> | Channel 1 Timer interrupt flag. |
| <i>kLPIT_Channel2TimerFlag</i> | Channel 2 Timer interrupt flag. |
| <i>kLPIT_Channel3TimerFlag</i> | Channel 3 Timer interrupt flag. |

**23.6 Function Documentation****23.6.1 void LPIT\_Init ( LPIT\_Type \* *base*, const lpit\_config\_t \* *config* )**

This function issues a software reset to reset all channels and registers except the Module Control register.

## Note

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

## Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | LPIT peripheral base address.                |
| <i>config</i> | Pointer to the user configuration structure. |

**23.6.2 void LPIT\_Deinit ( LPIT\_Type \* *base* )**

Parameters

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

### 23.6.3 void LPIT\_GetDefaultConfig ( *lpit\_config\_t* \* *config* )

The default values are:

```
* config->enableRunInDebug = false;
* config->enableRunInDoze = false;
*
```

Parameters

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

### 23.6.4 status\_t LPIT\_SetupChannel ( *LPIT\_Type* \* *base*, *lpit\_chnl\_t* *channel*, const *lpit\_chnl\_params\_t* \* *chnlSetup* )

This function sets up the operation mode to one of the options available in the enumeration [lpit\\_timer\\_modes\\_t](#). It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | LPIT peripheral base address.     |
| <i>channel</i>   | Channel that is being configured. |
| <i>chnlSetup</i> | Configuration parameters.         |

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

Parameters

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

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

Parameters

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

### 23.6.7 static uint32\_t LPIT\_GetEnabledInterrupts ( LPIT\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

### 23.6.8 static uint32\_t LPIT\_GetStatusFlags ( LPIT\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

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

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

## Parameters

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

### 23.6.10 static void LPIT\_SetTimerPeriod ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel*, uint32\_t *ticks* ) [inline], [static]

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

## Note

User can call the utility macros provided in fsl\_common.h to convert to ticks.

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | LPIT peripheral base address.   |
| <i>channel</i> | Timer channel number.           |
| <i>ticks</i>   | Timer period in units of ticks. |

### 23.6.11 static void LPIT\_SetTimerValue ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel*, uint32\_t *ticks* ) [inline], [static]

In the Dual 16-bit Periodic Counter mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

## Note

Set TVAL register to 0 or 1 is invalid in compare mode.

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | LPIT peripheral base address.   |
| <i>channel</i> | Timer channel number.           |
| <i>ticks</i>   | Timer period in units of ticks. |

### 23.6.12 static uint32\_t LPIT\_GetCurrentTimerCount ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in fsl\_common.h to convert ticks to microseconds or milliseconds.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | LPIT peripheral base address. |
| <i>channel</i> | Timer channel number.         |

Returns

Current timer counting value in ticks.

### 23.6.13 static void LPIT\_StartTimer ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel* ) [inline], [static]

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | LPIT peripheral base address. |
| <i>channel</i> | Timer channel number.         |

### 23.6.14 static void LPIT\_StopTimer ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel* ) [inline], [static]

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | LPIT peripheral base address. |
| <i>channel</i> | Timer channel number.         |

### 23.6.15 static void LPIT\_Reset( LPIT\_Type \* *base* ) [inline], [static]

This resets all channels and registers except the Module Control Register.

Parameters

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

## Chapter 24

# LPSPI: Low Power Serial Peripheral Interface

### 24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power Serial Peripheral Interface (LPSPI) module of MCUXpresso SDK devices.

### Modules

- [LPSPI CMSIS Driver](#)
- [LPSPI FreeRTOS Driver](#)
- [LPSPI Peripheral driver](#)
- [LPSPI eDMA Driver](#)

## 24.2 LPSPI Peripheral driver

### 24.2.1 Overview

This section describes the programming interface of the LPSPI Peripheral driver. The LPSPI driver configures LPSPI module, provides the functional and transactional interfaces to build the LPSPI application.

### 24.2.2 Function groups

#### 24.2.2.1 LPSPI Initialization and De-initialization

This function group initializes the default configuration structure for master and slave, initializes the LPSPI master with a master configuration, initializes the LPSPI slave with a slave configuration, and de-initializes the LPSPI module.

#### 24.2.2.2 LPSPI Basic Operation

This function group enables/disables the LPSPI module both interrupt and DMA, gets the data register address for the DMA transfer, sets master and slave, starts and stops the transfer, and so on.

#### 24.2.2.3 LPSPI Transfer Operation

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

#### 24.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

#### 24.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

### 24.2.3 Typical use case

#### 24.2.3.1 Master Operation

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

### 24.2.3.2 Slave Operation

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

## Data Structures

- struct `lpspi_master_config_t`  
*LPSPI master configuration structure. [More...](#)*
- struct `lpspi_slave_config_t`  
*LPSPI slave configuration structure. [More...](#)*
- struct `lpspi_transfer_t`  
*LPSPI master/slave transfer structure. [More...](#)*
- struct `lpspi_master_handle_t`  
*LPSPI master transfer handle structure used for transactional API. [More...](#)*
- struct `lpspi_slave_handle_t`  
*LPSPI slave transfer handle structure used for transactional API. [More...](#)*

## Macros

- #define `LPSPI_DUMMY_DATA` (0x00U)  
*LPSPI dummy data if no Tx data.*
- #define `SPI_RETRY_TIMES` 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*
- #define `LPSPI_MASTER_PCS_SHIFT` (4U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_MASTER_PCS_MASK` (0xF0U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_MASTER_WIDTH_SHIFT` (16U)  
*LPSPI master width shift macro, internal used.*
- #define `LPSPI_MASTER_WIDTH_MASK` (0x30000U)  
*LPSPI master width shift mask, internal used.*
- #define `LPSPI_SLAVE_PCS_SHIFT` (4U)  
*LPSPI slave PCS shift macro , internal used.*
- #define `LPSPI_SLAVE_PCS_MASK` (0xF0U)  
*LPSPI slave PCS shift macro , internal used.*

## Typedefs

- typedef void(\* `lpspi_master_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, `status_t` status, void \*userData)  
*Master completion callback function pointer type.*
- typedef void(\* `lpspi_slave_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, `status_t` status, void \*userData)  
*Slave completion callback function pointer type.*

## Enumerations

- enum {
   
kStatus\_LPSPI\_Busy = MAKE\_STATUS(kStatusGroup\_LPSPI, 0),
   
kStatus\_LPSPI\_Error = MAKE\_STATUS(kStatusGroup\_LPSPI, 1),
   
kStatus\_LPSPI\_Idle = MAKE\_STATUS(kStatusGroup\_LPSPI, 2),
   
kStatus\_LPSPI\_OutOfRange = MAKE\_STATUS(kStatusGroup\_LPSPI, 3),
   
kStatus\_LPSPI\_Timeout = MAKE\_STATUS(kStatusGroup\_LPSPI, 4) }
   
*Status for the LPSPI driver.*
  - enum \_lpspi\_flags {
   
kLPSPI\_TxDataRequestFlag = LPSPI\_SR\_TDF\_MASK,
   
kLPSPI\_RxDataReadyFlag = LPSPI\_SR\_RDF\_MASK,
   
kLPSPI\_WordCompleteFlag = LPSPI\_SR\_WCF\_MASK,
   
kLPSPI\_FrameCompleteFlag = LPSPI\_SR\_FCF\_MASK,
   
kLPSPI\_TransferCompleteFlag = LPSPI\_SR\_TCF\_MASK,
   
kLPSPI\_TransmitErrorFlag = LPSPI\_SR\_TEF\_MASK,
   
kLPSPI\_ReceiveErrorFlag = LPSPI\_SR\_REF\_MASK,
   
kLPSPI\_DataMatchFlag = LPSPI\_SR\_DMF\_MASK,
   
kLPSPI\_ModuleBusyFlag = LPSPI\_SR\_MBFI\_MASK,
   
kLPSPI\_AllStatusFlag }
   
*LPSPI status flags in SPIx\_SR register.*
  - enum \_lpspi\_interrupt\_enable {
   
kLPSPI\_TxInterruptEnable = LPSPI\_IER\_TDIE\_MASK,
   
kLPSPI\_RxInterruptEnable = LPSPI\_IER\_RDIE\_MASK,
   
kLPSPI\_WordCompleteInterruptEnable = LPSPI\_IER\_WCIE\_MASK,
   
kLPSPI\_FrameCompleteInterruptEnable = LPSPI\_IER\_FCIE\_MASK,
   
kLPSPI\_TransferCompleteInterruptEnable = LPSPI\_IER\_TCIE\_MASK,
   
kLPSPI\_TransmitErrorInterruptEnable = LPSPI\_IER\_TEIE\_MASK,
   
kLPSPI\_ReceiveErrorInterruptEnable = LPSPI\_IER\_REIE\_MASK,
   
kLPSPI\_DataMatchInterruptEnable = LPSPI\_IER\_DMIE\_MASK,
   
kLPSPI\_AllInterruptEnable }
   
*LPSPI interrupt source.*
  - enum \_lpspi\_dma\_enable {
   
kLPSPI\_TxDmaEnable = LPSPI\_DER\_TDDE\_MASK,
   
kLPSPI\_RxDmaEnable = LPSPI\_DER\_RDDE\_MASK }
  - enum lpspi\_master\_slave\_mode\_t {
   
kLPSPI\_Master = 1U,
   
kLPSPI\_Slave = 0U }
  - enum lpspi\_which\_pcs\_t {
   
kLPSPI\_Pcs0 = 0U,
   
kLPSPI\_Pcs1 = 1U,
   
kLPSPI\_Pcs2 = 2U,
   
kLPSPI\_Pcs3 = 3U }
- LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).*

- enum `lpspi_pcs_polarity_config_t` {
   
  `kLPSPI_PcsActiveHigh` = 1U,
   
  `kLPSPI_PcsActiveLow` = 0U }
   
*LPSPI Peripheral Chip Select (PCS) Polarity configuration.*
- enum `_lpspi_pcs_polarity` {
   
  `kLPSPI_Pcs0ActiveLow` = 1U << 0,
   
  `kLPSPI_Pcs1ActiveLow` = 1U << 1,
   
  `kLPSPI_Pcs2ActiveLow` = 1U << 2,
   
  `kLPSPI_Pcs3ActiveLow` = 1U << 3,
   
  `kLPSPI_PcsAllActiveLow` = 0xFU }
   
*LPSPI Peripheral Chip Select (PCS) Polarity.*
- enum `lpspi_clock_polarity_t` {
   
  `kLPSPI_ClockPolarityActiveHigh` = 0U,
   
  `kLPSPI_ClockPolarityActiveLow` = 1U }
   
*LPSPI clock polarity configuration.*
- enum `lpspi_clock_phase_t` {
   
  `kLPSPI_ClockPhaseFirstEdge` = 0U,
   
  `kLPSPI_ClockPhaseSecondEdge` = 1U }
   
*LPSPI clock phase configuration.*
- enum `lpspi_shift_direction_t` {
   
  `kLPSPI_MsbFirst` = 0U,
   
  `kLPSPI_LsbFirst` = 1U }
   
*LPSPI data shifter direction options.*
- enum `lpspi_host_request_select_t` {
   
  `kLPSPI_HostReqExtPin` = 0U,
   
  `kLPSPI_HostReqInternalTrigger` = 1U }
   
*LPSPI Host Request select configuration.*
- enum `lpspi_match_config_t` {
   
  `kLPSI_MatchDisabled` = 0x0U,
   
  `kLPSI_1stWordEqualsM0orM1` = 0x2U,
   
  `kLPSI_AnyWordEqualsM0orM1` = 0x3U,
   
  `kLPSI_1stWordEqualsM0and2ndWordEqualsM1` = 0x4U,
   
  `kLPSI_AnyWordEqualsM0andNxtWordEqualsM1` = 0x5U,
   
  `kLPSI_1stWordAndM1EqualsM0andM1` = 0x6U,
   
  `kLPSI_AnyWordAndM1EqualsM0andM1` = 0x7U }
   
*LPSPI Match configuration options.*
- enum `lpspi_pin_config_t` {
   
  `kLPSPI_SdiInSdoOut` = 0U,
   
  `kLPSPI_SdiInSdiOut` = 1U,
   
  `kLPSPI_SdoInSdoOut` = 2U,
   
  `kLPSPI_SdoInSdiOut` = 3U }
   
*LPSPI pin (SDO and SDI) configuration.*
- enum `lpspi_data_out_config_t` {
   
  `kLpspiDataOutRetained` = 0U,
   
  `kLpspiDataOutTristate` = 1U }
   
*LPSPI data output configuration.*
- enum `lpspi_pcs_function_config_t` {

- ```
kLPSPI_PcsAsCs = 0U,
kLPSPI_PcsAsData = 1U }
```

LPSPI cs function configuration.
- enum `lpspi_transfer_width_t` {

```
kLPSPI_SingleBitXfer = 0U,
kLPSPI_TwoBitXfer = 1U,
kLPSPI_FourBitXfer = 2U }
```

LPSPI transfer width configuration.
- enum `lpspi_delay_type_t` {

```
kLPSPI_PcsToSck = 1U,
kLPSPI_LastSckToPcs,
kLPSPI_BetweenTransfer }
```

LPSPI delay type selection.
- enum `_lpspi_transfer_config_flag_for_master` {

```
kLPSPI_MasterPcs0 = 0U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs1 = 1U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs2 = 2U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterPcs3 = 3U << LPSPI_MASTER_PCS_SHIFT,
kLPSPI_MasterWidth1 = 0U << LPSPI_MASTER_WIDTH_SHIFT,
kLPSPI_MasterWidth2 = 1U << LPSPI_MASTER_WIDTH_SHIFT,
kLPSPI_MasterWidth4 = 2U << LPSPI_MASTER_WIDTH_SHIFT,
kLPSPI_MasterPcsContinuous = 1U << 20,
kLPSPI_MasterByteSwap }
```

Use this enumeration for LPSPI master transfer configFlags.
- enum `_lpspi_transfer_config_flag_for_slave` {

```
kLPSPI_SlavePcs0 = 0U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs1 = 1U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs2 = 2U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlavePcs3 = 3U << LPSPI_SLAVE_PCS_SHIFT,
kLPSPI_SlaveByteSwap }
```

Use this enumeration for LPSPI slave transfer configFlags.
- enum `_lpspi_transfer_state` {

```
kLPSPI_Idle = 0x0U,
kLPSPI_Busy,
kLPSPI_Error }
```

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Variables

- volatile uint8_t `g_lpspiDummyData` []

Global variable for dummy data value setting.

Driver version

- #define `FSL_LPSPI_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 6)`)

LPSPI driver version.

Initialization and deinitialization

- void [LPSPI_MasterInit](#) (LPSPI_Type *base, const [lpspi_master_config_t](#) *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.
- void [LPSPI_MasterGetDefaultConfig](#) ([lpspi_master_config_t](#) *masterConfig)

Sets the [lpspi_master_config_t](#) structure to default values.
- void [LPSPI_SlaveInit](#) (LPSPI_Type *base, const [lpspi_slave_config_t](#) *slaveConfig)

LPSPI slave configuration.
- void [LPSPI_SlaveGetDefaultConfig](#) ([lpspi_slave_config_t](#) *slaveConfig)

Sets the [lpspi_slave_config_t](#) structure to default values.
- void [LPSPI_Deinit](#) (LPSPI_Type *base)

De-initializes the LPSPI peripheral.
- void [LPSPI_Reset](#) (LPSPI_Type *base)

Restores the LPSPI peripheral to reset state.
- uint32_t [LPSPIGetInstance](#) (LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.
- static void [LPSPI_Enable](#) (LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Status

- static uint32_t [LPSPI_GetStatusFlags](#) (LPSPI_Type *base)

Gets the LPSPI status flag state.
- static uint8_t [LPSPI_GetTxFifoSize](#) (LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.
- static uint8_t [LPSPI_GetRxFifoSize](#) (LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.
- static uint32_t [LPSPI_GetTxFifoCount](#) (LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.
- static uint32_t [LPSPI_GetRxFifoCount](#) (LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.
- static void [LPSPI_ClearStatusFlags](#) (LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

Interrupts

- static void [LPSPI_EnableInterrupts](#) (LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.
- static void [LPSPI_DisableInterrupts](#) (LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

DMA Control

- static void **LPSPI_EnableDMA** (LPSPI_Type *base, uint32_t mask)
Enables the LPSPI DMA request.
- static void **LPSPI_DisableDMA** (LPSPI_Type *base, uint32_t mask)
Disables the LPSPI DMA request.
- static uint32_t **LPSPI_GetTxRegisterAddress** (LPSPI_Type *base)
Gets the LPSPI Transmit Data Register address for a DMA operation.
- static uint32_t **LPSPI_GetRxRegisterAddress** (LPSPI_Type *base)
Gets the LPSPI Receive Data Register address for a DMA operation.

Bus Operations

- bool **LPSPI_CheckTransferArgument** (LPSPI_Type *base, lpspi_transfer_t *transfer, bool isEdma)
Check the argument for transfer .
- static void **LPSPI_SetMasterSlaveMode** (LPSPI_Type *base, lpspi_master_slave_mode_t mode)
Configures the LPSPI for either master or slave.
- static void **LPSPI_SelectTransferPCS** (LPSPI_Type *base, lpspi_which_pcs_t select)
Configures the peripheral chip select used for the transfer.
- static void **LPSPI_SetPCSContinous** (LPSPI_Type *base, bool IsContinous)
Set the PCS signal to continuous or uncontinuous mode.
- static bool **LPSPI_IsMaster** (LPSPI_Type *base)
Returns whether the LPSPI module is in master mode.
- static void **LPSPI_FlushFifo** (LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)
Flushes the LPSPI FIFOs.
- static void **LPSPI_SetFifoWatermarks** (LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)
Sets the transmit and receive FIFO watermark values.
- static void **LPSPI_SetAllPcsPolarity** (LPSPI_Type *base, uint32_t mask)
Configures all LPSPI peripheral chip select polarities simultaneously.
- static void **LPSPI_SetFrameSize** (LPSPI_Type *base, uint32_t frameSize)
Configures the frame size.
- uint32_t **LPSPI_MasterSetBaudRate** (LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)
Sets the LPSPI baud rate in bits per second.
- void **LPSPI_MasterSetDelayScaler** (LPSPI_Type *base, uint32_t scaler, lpspi_delay_type_t whichDelay)
Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).
- uint32_t **LPSPI_MasterSetDelayTimes** (LPSPI_Type *base, uint32_t delayTimeInNanoSec, lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz)
Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).
- static void **LPSPI_WriteData** (LPSPI_Type *base, uint32_t data)
Writes data into the transmit data buffer.
- static uint32_t **LPSPI_ReadData** (LPSPI_Type *base)
Reads data from the data buffer.
- void **LPSPI_SetDummyData** (LPSPI_Type *base, uint8_t dummyData)
Set up the dummy data.

Transactional

- void [LPSPI_MasterTransferCreateHandle](#) (LPSPI_Type *base, lpspi_master_handle_t *handle, lpspi_master_transfer_callback_t callback, void *userData)

Initializes the LPSPI master handle.
- status_t [LPSPI_MasterTransferBlocking](#) (LPSPI_Type *base, lpspi_transfer_t *transfer)

LPSPI master transfer data using a polling method.
- status_t [LPSPI_MasterTransferNonBlocking](#) (LPSPI_Type *base, lpspi_master_handle_t *handle, lpspi_transfer_t *transfer)

LPSPI master transfer data using an interrupt method.
- status_t [LPSPI_MasterTransferGetCount](#) (LPSPI_Type *base, lpspi_master_handle_t *handle, size_t *count)

Gets the master transfer remaining bytes.
- void [LPSPI_MasterTransferAbort](#) (LPSPI_Type *base, lpspi_master_handle_t *handle)

LPSPI master abort transfer which uses an interrupt method.
- void [LPSPI_MasterTransferHandleIRQ](#) (LPSPI_Type *base, lpspi_master_handle_t *handle)

LPSPI Master IRQ handler function.
- void [LPSPI_SlaveTransferCreateHandle](#) (LPSPI_Type *base, lpspi_slave_handle_t *handle, lpspi_slave_transfer_callback_t callback, void *userData)

Initializes the LPSPI slave handle.
- status_t [LPSPI_SlaveTransferNonBlocking](#) (LPSPI_Type *base, lpspi_slave_handle_t *handle, lpspi_transfer_t *transfer)

LPSPI slave transfer data using an interrupt method.
- status_t [LPSPI_SlaveTransferGetCount](#) (LPSPI_Type *base, lpspi_slave_handle_t *handle, size_t *count)

Gets the slave transfer remaining bytes.
- void [LPSPI_SlaveTransferAbort](#) (LPSPI_Type *base, lpspi_slave_handle_t *handle)

LPSPI slave aborts a transfer which uses an interrupt method.
- void [LPSPI_SlaveTransferHandleIRQ](#) (LPSPI_Type *base, lpspi_slave_handle_t *handle)

LPSPI Slave IRQ handler function.
- bool [LPSPI_WaitTxFifoEmpty](#) (LPSPI_Type *base)

Wait for tx FIFO to be empty.

24.2.4 Data Structure Documentation

24.2.4.1 struct lpspi_master_config_t

Data Fields

- uint32_t **baudRate**

Baud Rate for LPSPI.
- uint32_t **bitsPerFrame**

Bits per frame, minimum 8, maximum 4096.
- lpspi_clock_polarity_t **cpol**

Clock polarity.
- lpspi_clock_phase_t **cpha**

Clock phase.
- lpspi_shift_direction_t **direction**

- *MSB or LSB data shift direction.*
- `uint32_t pcsToSckDelayInNanoSec`
PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay.
- `uint32_t lastSckToPcsDelayInNanoSec`
Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay.
- `uint32_t betweenTransferDelayInNanoSec`
After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay.
- `lpspi_which_pcs_t whichPcs`
Desired Peripheral Chip Select (PCS).
- `lpspi_pcs_polarity_config_t pcsActiveHighOrLow`
Desired PCS active high or low.
- `lpspi_pin_config_t pinCfg`
Configures which pins are used for input and output data during single bit transfers.
- `lpspi_pcs_function_config_t pcsFunc`
Configures cs pins function.
- `lpspi_data_out_config_t dataOutConfig`
Configures if the output data is tristated between accesses (LPSPI_PCS is negated).
- `bool enableInputDelay`
Enable master to sample the input data on a delayed SCK.

Field Documentation

- (1) `uint32_t lpspi_master_config_t::baudRate`
- (2) `uint32_t lpspi_master_config_t::bitsPerFrame`
- (3) `lpspi_clock_polarity_t lpspi_master_config_t::cpol`
- (4) `lpspi_clock_phase_t lpspi_master_config_t::cpha`
- (5) `lpspi_shift_direction_t lpspi_master_config_t::direction`
- (6) `uint32_t lpspi_master_config_t::pcsToSckDelayInNanoSec`

It sets the boundary value if out of range.

- (7) `uint32_t lpspi_master_config_t::lastSckToPcsDelayInNanoSec`

It sets the boundary value if out of range.

- (8) `uint32_t lpspi_master_config_t::betweenTransferDelayInNanoSec`

It sets the boundary value if out of range.

- (9) `lpspi_which_pcs_t lpspi_master_config_t::whichPcs`
- (10) `lpspi_pin_config_t lpspi_master_config_t::pinCfg`
- (11) `lpspi_pcs_function_config_t lpspi_master_config_t::pcsFunc`
- (12) `lpspi_data_out_config_t lpspi_master_config_t::dataOutConfig`
- (13) `bool lpspi_master_config_t::enableInputDelay`

This can help improve slave setup time. Refer to device data sheet for specific time length.

24.2.4.2 struct lpspi_slave_config_t

Data Fields

- `uint32_t bitsPerFrame`
Bits per frame, minimum 8, maximum 4096.
- `lpspi_clock_polarity_t cpol`
Clock polarity.
- `lpspi_clock_phase_t cpha`
Clock phase.
- `lpspi_shift_direction_t direction`
MSB or LSB data shift direction.
- `lpspi_which_pcs_t whichPcs`
Desired Peripheral Chip Select (pcs)
- `lpspi_pcs_polarity_config_t pcsActiveHighOrLow`
Desired PCS active high or low.
- `lpspi_pin_config_t pinCfg`
Configures which pins are used for input and output data during single bit transfers.
- `lpspi_data_out_config_t dataOutConfig`
Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

Field Documentation

- (1) `uint32_t lpspi_slave_config_t::bitsPerFrame`
- (2) `lpspi_clock_polarity_t lpspi_slave_config_t::cpol`
- (3) `lpspi_clock_phase_t lpspi_slave_config_t::cpha`
- (4) `lpspi_shift_direction_t lpspi_slave_config_t::direction`
- (5) `lpspi_pin_config_t lpspi_slave_config_t::pinCfg`
- (6) `lpspi_data_out_config_t lpspi_slave_config_t::dataOutConfig`

24.2.4.3 struct lpspi_transfer_t

Data Fields

- `uint8_t * txData`
Send buffer.
- `uint8_t * rxData`
Receive buffer.
- `volatile size_t dataSize`
Transfer bytes.
- `uint32_t configFlags`
Transfer transfer configuration flags.

Field Documentation

- (1) `uint8_t* lpspi_transfer_t::txData`
- (2) `uint8_t* lpspi_transfer_t::rxData`
- (3) `volatile size_t lpspi_transfer_t::dataSize`
- (4) `uint32_t lpspi_transfer_t::configFlags`

Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

24.2.4.4 struct _lpspi_master_handle

Forward declaration of the `_lpspi_master_handle` typedefs.

Data Fields

- `volatile bool isPcsContinuous`
Is PCS continuous in transfer.
- `volatile bool writeTcrlInlsr`

- volatile bool **isByteSwap**
A flag that whether should write TCR in ISR.
- volatile bool **isTxMask**
A flag that whether should byte swap.
- volatile uint16_t **bytesPerFrame**
A flag that whether TCR[TXMSK] is set.
- volatile uint8_t **fifoSize**
Number of bytes in each frame.
- volatile uint8_t **rxWatermark**
FIFO dataSize.
- volatile uint8_t **bytesEachWrite**
Rx watermark.
- volatile uint8_t **bytesEachRead**
Bytes for each write TDR.
- uint8_t *volatile **txData**
Bytes for each read RDR.
- uint8_t *volatile **rxData**
Send buffer.
- uint8_t *volatile **txData**
Receive buffer.
- volatile size_t **txRemainingByteCount**
Number of bytes remaining to send.
- volatile size_t **rxRemainingByteCount**
Number of bytes remaining to receive.
- volatile uint32_t **writeRegRemainingTimes**
Write TDR register remaining times.
- volatile uint32_t **readRegRemainingTimes**
Read RDR register remaining times.
- uint32_t **totalByteCount**
Number of transfer bytes.
- uint32_t **txBuffIfNull**
Used if the txData is NULL.
- volatile uint8_t **state**
LPSPI transfer state , _lpspi_transfer_state.
- **lpspi_master_transfer_callback_t callback**
Completion callback.
- void * **userData**
Callback user data.

Field Documentation

- (1) volatile bool `lpspi_master_handle_t::isPcsContinuous`
- (2) volatile bool `lpspi_master_handle_t::writeTcrInIsr`
- (3) volatile bool `lpspi_master_handle_t::isByteSwap`
- (4) volatile bool `lpspi_master_handle_t::isTxMask`
- (5) volatile uint8_t `lpspi_master_handle_t::fifoSize`
- (6) volatile uint8_t `lpspi_master_handle_t::rxWatermark`
- (7) volatile uint8_t `lpspi_master_handle_t::bytesEachWrite`
- (8) volatile uint8_t `lpspi_master_handle_t::bytesEachRead`
- (9) uint8_t* volatile `lpspi_master_handle_t::txData`
- (10) uint8_t* volatile `lpspi_master_handle_t::rxData`
- (11) volatile size_t `lpspi_master_handle_t::txRemainingByteCount`
- (12) volatile size_t `lpspi_master_handle_t::rxRemainingByteCount`
- (13) volatile uint32_t `lpspi_master_handle_t::writeRegRemainingTimes`
- (14) volatile uint32_t `lpspi_master_handle_t::readRegRemainingTimes`
- (15) uint32_t `lpspi_master_handle_t::txBuffIfNull`
- (16) volatile uint8_t `lpspi_master_handle_t::state`
- (17) `lpspi_master_transfer_callback_t lpspi_master_handle_t::callback`
- (18) void* `lpspi_master_handle_t::userData`

24.2.4.5 struct _lpspi_slave_handle

Forward declaration of the `_lpspi_slave_handle` typedefs.

Data Fields

- volatile bool `isByteSwap`
A flag that whether should byte swap.
- volatile uint8_t `fifoSize`
FIFO dataSize.
- volatile uint8_t `rxWatermark`
Rx watermark.

- volatile uint8_t **bytesEachWrite**
Bytes for each write TDR.
- volatile uint8_t **bytesEachRead**
Bytes for each read RDR.
- uint8_t *volatile **txData**
Send buffer.
- uint8_t *volatile **rxData**
Receive buffer.
- volatile size_t **txRemainingByteCount**
Number of bytes remaining to send.
- volatile size_t **rxRemainingByteCount**
Number of bytes remaining to receive.
- volatile uint32_t **writeRegRemainingTimes**
Write TDR register remaining times.
- volatile uint32_t **readRegRemainingTimes**
Read RDR register remaining times.
- uint32_t **totalByteCount**
Number of transfer bytes.
- volatile uint8_t **state**
LPSPI transfer state , _lpspi_transfer_state.
- volatile uint32_t **errorCount**
Error count for slave transfer.
- **lpspi_slave_transfer_callback_t callback**
Completion callback.
- void * **userData**
Callback user data.

Field Documentation

- (1) volatile bool lpspi_slave_handle_t::isByteSwap
- (2) volatile uint8_t lpspi_slave_handle_t::fifoSize
- (3) volatile uint8_t lpspi_slave_handle_t::rxWatermark
- (4) volatile uint8_t lpspi_slave_handle_t::bytesEachWrite
- (5) volatile uint8_t lpspi_slave_handle_t::bytesEachRead
- (6) uint8_t* volatile lpspi_slave_handle_t::txData
- (7) uint8_t* volatile lpspi_slave_handle_t::rxData
- (8) volatile size_t lpspi_slave_handle_t::txRemainingByteCount
- (9) volatile size_t lpspi_slave_handle_t::rxRemainingByteCount
- (10) volatile uint32_t lpspi_slave_handle_t::writeRegRemainingTimes
- (11) volatile uint32_t lpspi_slave_handle_t::readRegRemainingTimes
- (12) volatile uint8_t lpspi_slave_handle_t::state
- (13) volatile uint32_t lpspi_slave_handle_t::errorCount
- (14) lpspi_slave_transfer_callback_t lpspi_slave_handle_t::callback
- (15) void* lpspi_slave_handle_t::userData

24.2.5 Macro Definition Documentation

24.2.5.1 #define FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 6, 6))

24.2.5.2 #define LPSPI_DUMMY_DATA (0x00U)

Dummy data used for tx if there is not txData.

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

24.2.5.4 `#define LPSPI_MASTER_PCS_SHIFT (4U)`

24.2.5.5 `#define LPSPI_MASTER_PCS_MASK (0xF0U)`

24.2.5.6 `#define LPSPI_SLAVE_PCS_SHIFT (4U)`

24.2.5.7 `#define LPSPI_SLAVE_PCS_MASK (0xF0U)`

24.2.6 Typedef Documentation

24.2.6.1 `typedef void(* lpspi_master_transfer_callback_t)(LPSPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)`

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	Pointer to the handle for the LPSPI master.
<i>status</i>	Success or error code describing whether the transfer is completed.
<i>userData</i>	Arbitrary pointer-dataSized value passed from the application.

24.2.6.2 `typedef void(* lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle, status_t status, void *userData)`

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	Pointer to the handle for the LPSPI slave.
<i>status</i>	Success or error code describing whether the transfer is completed.
<i>userData</i>	Arbitrary pointer-dataSized value passed from the application.

24.2.7 Enumeration Type Documentation

24.2.7.1 anonymous enum

Enumerator

kStatus_LPSPI_Busy LPSPI transfer is busy.
kStatus_LPSPI_Error LPSPI driver error.
kStatus_LPSPI_Idle LPSPI is idle.
kStatus_LPSPI_OutOfRange LPSPI transfer out Of range.
kStatus_LPSPI_Timeout LPSPI timeout polling status flags.

24.2.7.2 enum _lpspi_flags

Enumerator

kLPSPI_TxDataRequestFlag Transmit data flag.
kLPSPI_RxDataReadyFlag Receive data flag.
kLPSPI_WordCompleteFlag Word Complete flag.
kLPSPI_FrameCompleteFlag Frame Complete flag.
kLPSPI_TransferCompleteFlag Transfer Complete flag.
kLPSPI_TransmitErrorFlag Transmit Error flag (FIFO underrun)
kLPSPI_ReceiveErrorFlag Receive Error flag (FIFO overrun)

kLPSPI_DataMatchFlag Data Match flag.
kLPSPI_ModuleBusyFlag Module Busy flag.
kLPSPI_AllStatusFlag Used for clearing all w1c status flags.

24.2.7.3 enum _lpspi_interrupt_enable

Enumerator

kLPSPI_TxInterruptEnable Transmit data interrupt enable.
kLPSPI_RxInterruptEnable Receive data interrupt enable.
kLPSPI_WordCompleteInterruptEnable Word complete interrupt enable.
kLPSPI_FrameCompleteInterruptEnable Frame complete interrupt enable.
kLPSPI_TransferCompleteInterruptEnable Transfer complete interrupt enable.
kLPSPI_TransmitErrorInterruptEnable Transmit error interrupt enable(FIFO underrun)
kLPSPI_ReceiveErrorInterruptEnable Receive Error interrupt enable (FIFO overrun)
kLPSPI_DataMatchInterruptEnable Data Match interrupt enable.
kLPSPI_AllInterruptEnable All above interrupts enable.

24.2.7.4 enum _lpspi_dma_enable

Enumerator

kLPSPI_TxDmaEnable Transmit data DMA enable.
kLPSPI_RxDmaEnable Receive data DMA enable.

24.2.7.5 enum lpspi_master_slave_mode_t

Enumerator

kLPSPI_Master LPSPI peripheral operates in master mode.
kLPSPI_Slave LPSPI peripheral operates in slave mode.

24.2.7.6 enum lpspi_which_pcs_t

Enumerator

kLPSPI_Pcs0 PCS[0].
kLPSPI_Pcs1 PCS[1].
kLPSPI_Pcs2 PCS[2].
kLPSPI_Pcs3 PCS[3].

24.2.7.7 enum lpspi_pcs_polarity_config_t

Enumerator

kLPSPI_PcsActiveHigh PCS Active High (idles low)

kLPSPI_PcsActiveLow PCS Active Low (idles high)

24.2.7.8 enum _lpspi_pcs_polarity

Enumerator

kLPSPI_Pcs0ActiveLow Pcs0 Active Low (idles high).

kLPSPI_Pcs1ActiveLow Pcs1 Active Low (idles high).

kLPSPI_Pcs2ActiveLow Pcs2 Active Low (idles high).

kLPSPI_Pcs3ActiveLow Pcs3 Active Low (idles high).

kLPSPI_PcsAllActiveLow Pcs0 to Pcs5 Active Low (idles high).

24.2.7.9 enum lpspi_clock_polarity_t

Enumerator

kLPSPI_ClockPolarityActiveHigh CPOL=0. Active-high LPSPI clock (idles low)

kLPSPI_ClockPolarityActiveLow CPOL=1. Active-low LPSPI clock (idles high)

24.2.7.10 enum lpspi_clock_phase_t

Enumerator

kLPSPI_ClockPhaseFirstEdge CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

kLPSPI_ClockPhaseSecondEdge CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

24.2.7.11 enum lpspi_shift_direction_t

Enumerator

kLPSPI_MsbFirst Data transfers start with most significant bit.

kLPSPI_LsbFirst Data transfers start with least significant bit.

24.2.7.12 enum lpspi_host_request_select_t

Enumerator

kLPSPI_HostReqExtPin Host Request is an ext pin.

kLPSPI_HostReqInternalTrigger Host Request is an internal trigger.

24.2.7.13 enum lpspi_match_config_t

Enumerator

kLPSI_MatchDisabled LPSPI Match Disabled.

kLPSI_1stWordEqualsM0orM1 LPSPI Match Enabled.

kLPSI_AnyWordEqualsM0orM1 LPSPI Match Enabled.

kLPSI_1stWordEqualsM0and2ndWordEqualsM1 LPSPI Match Enabled.

kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 LPSPI Match Enabled.

kLPSI_1stWordAndM1EqualsM0andM1 LPSPI Match Enabled.

kLPSI_AnyWordAndM1EqualsM0andM1 LPSPI Match Enabled.

24.2.7.14 enum lpspi_pin_config_t

Enumerator

kLPSPI_SdiInSdoOut LPSPI SDI input, SDO output.

kLPSPI_SdiInSdiOut LPSPI SDI input, SDI output.

kLPSPI_SdoInSdoOut LPSPI SDO input, SDO output.

kLPSPI_SdoInSdiOut LPSPI SDO input, SDI output.

24.2.7.15 enum lpspi_data_out_config_t

Enumerator

kLpspiDataOutRetained Data out retains last value when chip select is de-asserted.

kLpspiDataOutTristate Data out is tristated when chip select is de-asserted.

24.2.7.16 enum lpspi_pcs_function_config_t

Enumerator

kLPSPI_PcsAsCs PCS pin select as cs function.

kLPSPI_PcsAsData PCS pin select as date function.

24.2.7.17 enum lpspi_transfer_width_t

Enumerator

kLPSPI_SingleBitXfer 1-bit shift at a time, data out on SDO, in on SDI (normal mode)

kLPSPI_TwoBitXfer 2-bits shift out on SDO/SDI and in on SDO/SDI

kLPSPI_FourBitXfer 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

24.2.7.18 enum lpspi_delay_type_t

Enumerator

kLPSPI_PcsToSck PCS-to-SCK delay.

kLPSPI_LastSckToPcs Last SCK edge to PCS delay.

kLPSPI_BetweenTransfer Delay between transfers.

24.2.7.19 enum _lpspi_transfer_config_flag_for_master

Enumerator

kLPSPI_MasterPcs0 LPSPI master transfer use PCS0 signal.

kLPSPI_MasterPcs1 LPSPI master transfer use PCS1 signal.

kLPSPI_MasterPcs2 LPSPI master transfer use PCS2 signal.

kLPSPI_MasterPcs3 LPSPI master transfer use PCS3 signal.

kLPSPI_MasterWidth1 LPSPI master transfer 1bit.

kLPSPI_MasterWidth2 LPSPI master transfer 2bit.

kLPSPI_MasterWidth4 LPSPI master transfer 4bit.

kLPSPI_MasterPcsContinuous Is PCS signal continuous.

kLPSPI_MasterByteSwap Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

1. If you set bitPerFrame = 8 , no matter the *kLPSPI_MasterByteSwap* flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the *kLPSPI_MasterByteSwap* flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the *kLPSPI_MasterByteSwap* flag.
3. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the *kLPSPI_MasterByteSwap* flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the *kLPSPI_MasterByteSwap* flag.

24.2.7.20 enum _lpspi_transfer_config_flag_for_slave

Enumerator

kLPSPI_SlavePcs0 LPSPI slave transfer use PCS0 signal.

kLPSPI_SlavePcs1 LPSPI slave transfer use PCS1 signal.

kLPSPI_SlavePcs2 LPSPI slave transfer use PCS2 signal.

kLPSPI_SlavePcs3 LPSPI slave transfer use PCS3 signal.

kLPSPI_SlaveByteSwap Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

1. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.
3. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

24.2.7.21 enum _lpspi_transfer_state

Enumerator

kLPSPI_Idle Nothing in the transmitter/receiver.

kLPSPI_Busy Transfer queue is not finished.

kLPSPI_Error Transfer error.

24.2.8 Function Documentation

24.2.8.1 void LPSPI_MasterInit (**LPSPI_Type** * *base*, const **lpspi_master_config_t** * *masterConfig*, **uint32_t** *srcClock_Hz*)

Parameters

<i>base</i>	LPSPI peripheral address.
<i>masterConfig</i>	Pointer to structure lpspi_master_config_t .
<i>srcClock_Hz</i>	Module source input clock in Hertz

24.2.8.2 void LPSPI_MasterGetDefaultConfig (**lpspi_master_config_t** * *masterConfig*)

This API initializes the configuration structure for [LPSPI_MasterInit\(\)](#). The initialized structure can remain unchanged in [LPSPI_MasterInit\(\)](#), or can be modified before calling the [LPSPI_MasterInit\(\)](#). Example:

```
* lpspi_master_config_t masterConfig;
* LPSPI_MasterGetDefaultConfig(&masterConfig);
*
```

Parameters

<i>masterConfig</i>	pointer to <code>lpspi_master_config_t</code> structure
---------------------	---

24.2.8.3 void LPSPI_SlavelInit (`LPSPI_Type` * *base*, `const lpspi_slave_config_t` * *slaveConfig*)

Parameters

<i>base</i>	LPSPI peripheral address.
<i>slaveConfig</i>	Pointer to a structure <code>lpspi_slave_config_t</code> .

24.2.8.4 void LPSPI_SlaveGetDefaultConfig (`lpspi_slave_config_t` * *slaveConfig*)

This API initializes the configuration structure for `LPSPI_SlaveInit()`. The initialized structure can remain unchanged in `LPSPI_SlaveInit()` or can be modified before calling the `LPSPI_SlaveInit()`. Example:

```
*     lpspi_slave_config_t slaveConfig;
*     LPSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

<i>slaveConfig</i>	pointer to <code>lpspi_slave_config_t</code> structure.
--------------------	---

24.2.8.5 void LPSPI_Deinit (`LPSPI_Type` * *base*)

Call this API to disable the LPSPI clock.

Parameters

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

24.2.8.6 void LPSPI_Reset (`LPSPI_Type` * *base*)

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

Parameters

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

24.2.8.7 `uint32_t LPSPI_GetInstance(LPSPI_Type * base)`

Parameters

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

Returns

LPSPI instance.

24.2.8.8 `static void LPSPI_Enable(LPSPI_Type * base, bool enable) [inline], [static]`

Parameters

<i>base</i>	LPSPI peripheral address.
<i>enable</i>	Pass true to enable module, false to disable module.

24.2.8.9 `static uint32_t LPSPI_GetStatusFlags(LPSPI_Type * base) [inline], [static]`

Parameters

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

Returns

The LPSPI status(in SR register).

24.2.8.10 `static uint8_t LPSPI_GetTxFifoSize(LPSPI_Type * base) [inline], [static]`

Parameters

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

Returns

The LPSPI Tx FIFO size.

24.2.8.11 static uint8_t LPSPI_GetRxFifoSize (LPSPI_Type * *base*) [inline], [static]

Parameters

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

Returns

The LPSPI Rx FIFO size.

24.2.8.12 static uint32_t LPSPI_GetTxFifoCount (LPSPI_Type * *base*) [inline], [static]

Parameters

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

Returns

The number of words in the transmit FIFO.

24.2.8.13 static uint32_t LPSPI_GetRxFifoCount (LPSPI_Type * *base*) [inline], [static]

Parameters

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

Returns

The number of words in the receive FIFO.

24.2.8.14 static void LPSPI_ClearStatusFlags (LPSPI_Type * *base*, uint32_t *statusFlags*) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
* LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag |
    kLPSPI_RxDataReadyFlag);
*
```

Parameters

<i>base</i>	LPSPI peripheral address.
<i>statusFlags</i>	The status flag used from type _lpspi_flags.

< The status flags are cleared by writing 1 (w1c).

24.2.8.15 static void LPSPI_EnableInterrupts (LPSPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
* LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable |
    kLPSPI_RxInterruptEnable );
*
```

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mask</i>	The interrupt mask; Use the enum _lpspi_interrupt_enable.

24.2.8.16 static void LPSPI_DisableInterrupts (LPSPI_Type * *base*, uint32_t *mask*) [inline], [static]

```
* LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable |
    kLPSPI_RxInterruptEnable );
*
```

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mask</i>	The interrupt mask; Use the enum _lpspi_interrupt_enable.

24.2.8.17 static void LPSPI_EnableDMA (LPSPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
*   LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable |
                    kLPSPI_Rx_dmaEnable);
*
```

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mask</i>	The interrupt mask; Use the enum _lpspi_dma_enable.

24.2.8.18 static void LPSPI_DisableDMA (LPSPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
*   SPI_DisableDMA(base, kLPSPI_TxDmaEnable |
                    kLPSPI_Rx_dmaEnable);
*
```

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mask</i>	The interrupt mask; Use the enum _lpspi_dma_enable.

24.2.8.19 static uint32_t LPSPI_GetTxRegisterAddress (LPSPI_Type * *base*) [inline], [static]

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

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

Returns

The LPSPI Transmit Data Register address.

24.2.8.20 static uint32_t LPSPI_GetRxRegisterAddress (LPSPI_Type * *base*) [inline], [static]

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

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

Returns

The LPSPI Receive Data Register address.

24.2.8.21 bool LPSPI_CheckTransferArgument (LPSPI_Type * *base*, lpspi_transfer_t * *transfer*, bool *isEdma*)

Parameters

<i>base</i>	LPSPI peripheral address.
<i>transfer</i>	the transfer struct to be used.
<i>isEdma</i>	True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns

Return true for right and false for wrong.

24.2.8.22 static void LPSPI_SetMasterSlaveMode (LPSPI_Type * *base*, lpspi_master_slave_mode_t *mode*) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mode</i>	Mode setting (master or slave) of type lpspi_master_slave_mode_t.

24.2.8.23 static void LPSPI_SelectTransferPCS (LPSPI_Type * *base*, lpspi_which_pcs_t *select*) [inline], [static]

Parameters

<i>base</i>	LPSPI peripheral address.
<i>select</i>	LPSPI Peripheral Chip Select (PCS) configuration.

24.2.8.24 static void LPSPI_SetPCSContinuous (LPSPI_Type * *base*, bool *IsContinuous*) [inline], [static]

Note

In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

<i>base</i>	LPSPI peripheral address.
<i>IsContinuous</i>	True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

24.2.8.25 static bool LPSPI_IsMaster (LPSPI_Type * *base*) [inline], [static]

Parameters

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

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

24.2.8.26 static void LPSPI_FlushFifo (LPSPI_Type * *base*, bool *flushTxFifo*, bool *flushRxFifo*) [inline], [static]

Parameters

<i>base</i>	LPSPI peripheral address.
<i>flushTxFifo</i>	Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
<i>flushRxFifo</i>	Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

24.2.8.27 static void LPSPI_SetFifoWatermarks (LPSPI_Type * *base*, uint32_t *txWater*, uint32_t *rxWater*) [inline], [static]

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>txWater</i>	The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
<i>rxWater</i>	The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

24.2.8.28 static void LPSPI_SetAllPcsPolarity (LPSPI_Type * *base*, uint32_t *mask*) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
* LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow |
    kLPSPI_Pcs1ActiveLow);
*
```

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mask</i>	The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

24.2.8.29 static void LPSPI_SetFrameSize (LPSPI_Type * *base*, uint32_t *frameSize*) [inline], [static]

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>frameSize</i>	The frame size in number of bits.

24.2.8.30 uint32_t LPSPI_MasterSetBaudRate (LPSPI_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*, uint32_t * *tcrPrescaleValue*)

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale *tcrPrescaleValue* parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>baudRate_Bps</i>	The desired baud rate in bits per second.
<i>srcClock_Hz</i>	Module source input clock in Hertz.

<i>tcrPrescale-Value</i>	The TCR prescale value needed to program the TCR.
--------------------------	---

Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

24.2.8.31 void LPSPI_MasterSetDelayScaler (*LPSPI_Type * base, uint32_t scaler, lpspi_delay_type_t whichDelay*)

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type *lpspi_delay_type_t*.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>scaler</i>	The 8-bit delay value 0x00 to 0xFF (255).
<i>whichDelay</i>	The desired delay to configure, must be of type <i>lpspi_delay_type_t</i> .

24.2.8.32 uint32_t LPSPI_MasterSetDelayTimes (*LPSPI_Type * base, uint32_t delayTimeInNanoSec, lpspi_delay_type_t whichDelay, uint32_t srcClock_Hz*)

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type *lpspi_delay_type_t*.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the *delayTime* = *LPSPI_clockSource / (PRESCALE * Delay_scaler)*.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>delayTimeIn-NanoSec</i>	The desired delay value in nano-seconds.
<i>whichDelay</i>	The desired delay to configuration, which must be of type lpspi_delay_type_t.
<i>srcClock_Hz</i>	Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

24.2.8.33 static void LPSPI_WriteData (LPSPI_Type * *base*, uint32_t *data*) [inline], [static]

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>data</i>	The data word to be sent.

24.2.8.34 static uint32_t LPSPI_ReadData (LPSPI_Type * *base*) [inline], [static]

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

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

Returns

The data read from the data buffer.

24.2.8.35 void LPSPI_SetDummyData (LPSPI_Type * *base*, uint8_t *dummyData*)

Parameters

<i>base</i>	LPSPI peripheral address.
<i>dummyData</i>	Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

24.2.8.36 void LPSPI_MasterTransferCreateHandle (**LPSPI_Type * *base*,
lpspi_master_handle_t * *handle*, **lpspi_master_transfer_callback_t** *callback*,
void * *userData*)**

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

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	LPSPI handle pointer to lpspi_master_handle_t .
<i>callback</i>	DSPI callback.
<i>userData</i>	callback function parameter.

24.2.8.37 status_t LPSPI_MasterTransferBlocking (**LPSPI_Type * *base*, **lpspi_transfer_t** * *transfer*)**

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>transfer</i>	pointer to lpspi_transfer_t structure.

Returns

status of **status_t**.

24.2.8.38 status_t LPSPI_MasterTransferNonBlocking (**LPSPI_Type** * *base*, **lpspi_master_handle_t** * *handle*, **lpspi_transfer_t** * *transfer*)

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	pointer to lpspi_master_handle_t structure which stores the transfer state.
<i>transfer</i>	pointer to lpspi_transfer_t structure.

Returns

status of status_t.

24.2.8.39 status_t LPSPI_MasterTransferGetCount (**LPSPI_Type** * *base*, **lpspi_master_handle_t** * *handle*, **size_t** * *count*)

This function gets the master transfer remaining bytes.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	pointer to lpspi_master_handle_t structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

24.2.8.40 void LPSPI_MasterTransferAbort (**LPSPI_Type** * *base*, **lpspi_master_handle_t** * *handle*)

This function aborts a transfer which uses an interrupt method.

Parameters

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

24.2.8.41 void LPSPI_MasterTransferHandleIRQ (LPSPI_Type * *base*, lpspi_master_handle_t * *handle*)

This function processes the LPSPI transmit and receive IRQ.

Parameters

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

24.2.8.42 void LPSPI_SlaveTransferCreateHandle (LPSPI_Type * *base*, lpspi_slave_handle_t * *handle*, lpspi_slave_transfer_callback_t *callback*, void * *userData*)

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

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	LPSPI handle pointer to lpspi_slave_handle_t.
<i>callback</i>	DSPI callback.
<i>userData</i>	callback function parameter.

24.2.8.43 status_t LPSPI_SlaveTransferNonBlocking (LPSPI_Type * *base*, lpspi_slave_handle_t * *handle*, lpspi_transfer_t * *transfer*)

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state.
<i>transfer</i>	pointer to <code>lpspi_transfer_t</code> structure.

Returns

status of `status_t`.

24.2.8.44 `status_t LPSPI_SlaveTransferGetCount (LPSPI_Type * base, lpspi_slave_handle_t * handle, size_t * count)`

This function gets the slave transfer remaining bytes.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Returns

status of `status_t`.

24.2.8.45 `void LPSPI_SlaveTransferAbort (LPSPI_Type * base, lpspi_slave_handle_t * handle)`

This function aborts a transfer which uses an interrupt method.

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state.

24.2.8.46 `void LPSPI_SlaveTransferHandleIRQ (LPSPI_Type * base, lpspi_slave_handle_t * handle)`

This function processes the LPSPI transmit and receives an IRQ.

Parameters

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

24.2.8.47 bool LPSPI_WaitTxFifoEmpty (LPSPI_Type * *base*)

This function wait the tx fifo empty

Parameters

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

Returns

true for the tx FIFO is ready, false is not.

24.2.9 Variable Documentation

24.2.9.1 volatile uint8_t g_lpspiDummyData[]

24.3 LPSPI eDMA Driver

24.3.1 Overview

Data Structures

- struct `lpspi_master_edma_handle_t`
LPSPI master eDMA transfer handle structure used for transactional API. [More...](#)
- struct `lpspi_slave_edma_handle_t`
LPSPI slave eDMA transfer handle structure used for transactional API. [More...](#)

TypeDefs

- typedef void(* `lpspi_master_edma_transfer_callback_t`)(LPSPI_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)
Completion callback function pointer type.
- typedef void(* `lpspi_slave_edma_transfer_callback_t`)(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, status_t status, void *userData)
Completion callback function pointer type.

Functions

- void `LPSPI_MasterTransferCreateHandleEDMA` (LPSPI_Type *base, lpspi_master_edma_handle_t *handle, `lpspi_master_edma_transfer_callback_t` callback, void *userData, `edma_handle_t` *edmaRxRegToRxDataHandle, `edma_handle_t` *edmaTxDataToTxRegHandle)
Initializes the LPSPI master eDMA handle.
- status_t `LPSPI_MasterTransferEDMA` (LPSPI_Type *base, lpspi_master_edma_handle_t *handle, `lpspi_transfer_t` *transfer)
LPSPI master transfer data using eDMA.
- status_t `LPSPI_MasterTransferPrepareEDMALite` (LPSPI_Type *base, lpspi_master_edma_handle_t *handle, uint32_t configFlags)
LPSPI master config transfer parameter while using eDMA.
- status_t `LPSPI_MasterTransferEDMALite` (LPSPI_Type *base, lpspi_master_edma_handle_t *handle, `lpspi_transfer_t` *transfer)
LPSPI master transfer data using eDMA without configs.
- void `LPSPI_MasterTransferAbortEDMA` (LPSPI_Type *base, lpspi_master_edma_handle_t *handle)
LPSPI master aborts a transfer which is using eDMA.
- status_t `LPSPI_MasterTransferGetCountEDMA` (LPSPI_Type *base, lpspi_master_edma_handle_t *handle, size_t *count)
Gets the master eDMA transfer remaining bytes.
- void `LPSPI_SlaveTransferCreateHandleEDMA` (LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, `lpspi_slave_edma_transfer_callback_t` callback, void *userData, `edma_handle_t` *edmaRxRegToRxDataHandle, `edma_handle_t` *edmaTxDataToTxRegHandle)
Initializes the LPSPI slave eDMA handle.

- `status_t LPSPI_SlaveTransferEDMA` (`LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, lpspi_transfer_t *transfer)`
LPSPI slave transfers data using eDMA.
- `void LPSPI_SlaveTransferAbortEDMA` (`LPSPI_Type *base, lpspi_slave_edma_handle_t *handle)`
LPSPI slave aborts a transfer which is using eDMA.
- `status_t LPSPI_SlaveTransferGetCountEDMA` (`LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, size_t *count)`
Gets the slave eDMA transfer remaining bytes.

Driver version

- `#define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 2))`
LPSPI EDMA driver version.

24.3.2 Data Structure Documentation

24.3.2.1 struct _lpspi_master_edma_handle

Forward declaration of the `_lpspi_master_edma_handle` typedefs.

Data Fields

- `volatile bool isPcsContinuous`
Is PCS continuous in transfer.
- `volatile bool isByteSwap`
A flag that whether should byte swap.
- `volatile uint8_t fifoSize`
FIFO dataSize.
- `volatile uint8_t rxWatermark`
Rx watermark.
- `volatile uint8_t bytesEachWrite`
Bytes for each write TDR.
- `volatile uint8_t bytesEachRead`
Bytes for each read RDR.
- `volatile uint8_t bytesLastRead`
Bytes for last read RDR.
- `volatile bool isThereExtraRxBytes`
Is there extra RX byte.
- `uint8_t *volatile txData`
Send buffer.
- `uint8_t *volatile rxData`
Receive buffer.
- `volatile size_t txRemainingByteCount`
Number of bytes remaining to send.
- `volatile size_t rxRemainingByteCount`
Number of bytes remaining to receive.
- `volatile uint32_t writeRegRemainingTimes`

- volatile uint32_t **readRegRemainingTimes**
Write TDR register remaining times.
- uint32_t **totalByteCount**
Read RDR register remaining times.
- uint32_t **txBuffIfNull**
Number of transfer bytes.
- uint32_t **txBuffIfNull**
Used if there is not txData for DMA purpose.
- uint32_t **rxBuffIfNull**
Used if there is not rxData for DMA purpose.
- uint32_t **transmitCommand**
Used to write TCR for DMA purpose.
- volatile uint8_t **state**
LPSPI transfer state , _lpspi_transfer_state.
- uint8_t **nbytes**
eDMA minor byte transfer count initially configured.
- **lpspi_master_edma_transfer_callback_t callback**
Completion callback.
- void * **userData**
Callback user data.
- **edma_handle_t * edmaRxRegToRxDataHandle**
edma_handle_t handle point used for RxReg to RxData buff
- **edma_handle_t * edmaTxDataToTxRegHandle**
edma_handle_t handle point used for TxData to TxReg buff
- **edma_tcd_t lpspiSoftwareTCD [3]**
SoftwareTCD, internal used.

Field Documentation

- (1) volatile bool `lpspi_master_edma_handle_t::isPcsContinuous`
- (2) volatile bool `lpspi_master_edma_handle_t::isByteSwap`
- (3) volatile uint8_t `lpspi_master_edma_handle_t::fifoSize`
- (4) volatile uint8_t `lpspi_master_edma_handle_t::rxWatermark`
- (5) volatile uint8_t `lpspi_master_edma_handle_t::bytesEachWrite`
- (6) volatile uint8_t `lpspi_master_edma_handle_t::bytesEachRead`
- (7) volatile uint8_t `lpspi_master_edma_handle_t::bytesLastRead`
- (8) volatile bool `lpspi_master_edma_handle_t::isThereExtraRxBytes`
- (9) uint8_t* volatile `lpspi_master_edma_handle_t::txData`
- (10) uint8_t* volatile `lpspi_master_edma_handle_t::rxData`
- (11) volatile size_t `lpspi_master_edma_handle_t::txRemainingByteCount`
- (12) volatile size_t `lpspi_master_edma_handle_t::rxRemainingByteCount`
- (13) volatile uint32_t `lpspi_master_edma_handle_t::writeRegRemainingTimes`
- (14) volatile uint32_t `lpspi_master_edma_handle_t::readRegRemainingTimes`
- (15) uint32_t `lpspi_master_edma_handle_t::txBuffIfNull`
- (16) uint32_t `lpspi_master_edma_handle_t::rxBuffIfNull`
- (17) uint32_t `lpspi_master_edma_handle_t::transmitCommand`
- (18) volatile uint8_t `lpspi_master_edma_handle_t::state`
- (19) uint8_t `lpspi_master_edma_handle_t::nbytes`
- (20) `lpspi_master_edma_transfer_callback_t lpspi_master_edma_handle_t::callback`
- (21) void* `lpspi_master_edma_handle_t::userData`

24.3.2.2 struct _lpspi_slave_edma_handle

Forward declaration of the `_lpspi_slave_edma_handle` typedefs.

Data Fields

- volatile bool **isByteSwap**
A flag that whether should byte swap.
- volatile uint8_t **fifoSize**
FIFO dataSize.
- volatile uint8_t **rxWatermark**
Rx watermark.
- volatile uint8_t **bytesEachWrite**
Bytes for each write TDR.
- volatile uint8_t **bytesEachRead**
Bytes for each read RDR.
- volatile uint8_t **bytesLastRead**
Bytes for last read RDR.
- volatile bool **isThereExtraRxBytes**
Is there extra RX byte.
- uint8_t **nbytes**
eDMA minor byte transfer count initially configured.
- uint8_t *volatile **txData**
Send buffer.
- uint8_t *volatile **rxData**
Receive buffer.
- volatile size_t **txRemainingByteCount**
Number of bytes remaining to send.
- volatile size_t **rxRemainingByteCount**
Number of bytes remaining to receive.
- volatile uint32_t **writeRegRemainingTimes**
Write TDR register remaining times.
- volatile uint32_t **readRegRemainingTimes**
Read RDR register remaining times.
- uint32_t **totalByteCount**
Number of transfer bytes.
- uint32_t **txBuffIfNull**
Used if there is not txData for DMA purpose.
- uint32_t **rxBuffIfNull**
Used if there is not rxData for DMA purpose.
- volatile uint8_t **state**
LPSPI transfer state.
- uint32_t **errorCount**
Error count for slave transfer.
- **lpspi_slave_edma_transfer_callback_t callback**
Completion callback.
- void * **userData**
Callback user data.
- edma_handle_t * **edmaRxRegToRxDataHandle**
edma_handle_t handle point used for RxReg to RxData buff
- edma_handle_t * **edmaTxDataToTxRegHandle**
edma_handle_t handle point used for TxData to TxReg
- **edma_tcd_t lpspiSoftwareTCD [2]**
SoftwareTCD, internal used.

Field Documentation

- (1) volatile bool lpspi_slave_edma_handle_t::isByteSwap
- (2) volatile uint8_t lpspi_slave_edma_handle_t::fifoSize
- (3) volatile uint8_t lpspi_slave_edma_handle_t::rxWatermark
- (4) volatile uint8_t lpspi_slave_edma_handle_t::bytesEachWrite
- (5) volatile uint8_t lpspi_slave_edma_handle_t::bytesEachRead
- (6) volatile uint8_t lpspi_slave_edma_handle_t::bytesLastRead
- (7) volatile bool lpspi_slave_edma_handle_t::isThereExtraRxBytes
- (8) uint8_t lpspi_slave_edma_handle_t::nbytes
- (9) uint8_t* volatile lpspi_slave_edma_handle_t::txData
- (10) uint8_t* volatile lpspi_slave_edma_handle_t::rxData
- (11) volatile size_t lpspi_slave_edma_handle_t::txRemainingByteCount
- (12) volatile size_t lpspi_slave_edma_handle_t::rxRemainingByteCount
- (13) volatile uint32_t lpspi_slave_edma_handle_t::writeRegRemainingTimes
- (14) volatile uint32_t lpspi_slave_edma_handle_t::readRegRemainingTimes
- (15) uint32_t lpspi_slave_edma_handle_t::txBuffIfNull
- (16) uint32_t lpspi_slave_edma_handle_t::rxBuffIfNull
- (17) volatile uint8_t lpspi_slave_edma_handle_t::state
- (18) uint32_t lpspi_slave_edma_handle_t::errorCount
- (19) lpspi_slave_edma_transfer_callback_t lpspi_slave_edma_handle_t::callback
- (20) void* lpspi_slave_edma_handle_t::userData

24.3.3 Macro Definition Documentation**24.3.3.1 #define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 2))****24.3.4 Typedef Documentation****24.3.4.1 typedef void(* lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base,
lpspi_master_edma_handle_t *handle, status_t status, void *userData)**

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	Pointer to the handle for the LPSPI master.
<i>status</i>	Success or error code describing whether the transfer completed.
<i>userData</i>	Arbitrary pointer-dataSized value passed from the application.

24.3.4.2 **typedef void(* lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_edma_handle_t *handle, status_t status, void *userData)**

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	Pointer to the handle for the LPSPI slave.
<i>status</i>	Success or error code describing whether the transfer completed.
<i>userData</i>	Arbitrary pointer-dataSized value passed from the application.

24.3.5 Function Documentation

24.3.5.1 **void LPSPI_MasterTransferCreateHandleEDMA (LPSPI_Type * *base*, lpspi_master_edma_handle_t * *handle*, lpspi_master_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *edmaRxRegToRxDataHandle*, edma_handle_t * *edmaTxDataToTxRegHandle*)**

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

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

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

<i>handle</i>	LPSPI handle pointer to <code>lpspi_master_edma_handle_t</code> .
<i>callback</i>	LPSPI callback.
<i>userData</i>	callback function parameter.
<i>edmaRxRegTo-RxDataHandle</i>	edmaRxRegToRxDataHandle pointer to <code>edma_handle_t</code> .
<i>edmaTxData-ToTxReg-Handle</i>	edmaTxDataToTxRegHandle pointer to <code>edma_handle_t</code> .

24.3.5.2 `status_t LPSPI_MasterTransferEDMA (LPSPI_Type * base, lpspi_master_edma_handle_t * handle, lpspi_transfer_t * transfer)`

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state.
<i>transfer</i>	pointer to <code>lpspi_transfer_t</code> structure.

Returns

status of `status_t`.

24.3.5.3 `status_t LPSPI_MasterTransferPrepareEDMALite (LPSPI_Type * base, lpspi_master_edma_handle_t * handle, uint32_t configFlags)`

This function is preparing to transfer data using eDMA, work with `LPSPI_MasterTransferEDMALite`.

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state.
<i>configFlags</i>	transfer configuration flags. <code>_lpspi_transfer_config_flag_for_master</code> .

Returns

Indicates whether LPSPI master transfer was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully.
<i>kStatus_LPSPI_Busy</i>	The LPSPI device is busy.

24.3.5.4 `status_t LPSPI_MasterTransferEDMALite (LPSPI_Type * base, lpspi_master_edma_handle_t * handle, lpspi_transfer_t * transfer)`

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call `LPSPI_MasterTransferPrepareEDMALite` to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state.
<i>transfer</i>	pointer to <code>lpspi_transfer_t</code> structure, config field is not used.

Returns

Indicates whether LPSPI master transfer was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully.
<i>kStatus_LPSPI_Busy</i>	The LPSPI device is busy.
<i>kStatus_InvalidArgument</i>	The transfer structure is invalid.

24.3.5.5 void LPSPI_MasterTransferAbortEDMA (**LPSPI_Type** * *base*, **lpspi_master_edma_handle_t** * *handle*)

This function aborts a transfer which is using eDMA.

Parameters

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

24.3.5.6 status_t LPSPI_MasterTransferGetCountEDMA (**LPSPI_Type** * *base*, **lpspi_master_edma_handle_t** * *handle*, **size_t** * *count*)

This function gets the master eDMA transfer remaining bytes.

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the EDMA transaction.

Returns

status of **status_t**.

24.3.5.7 void LPSPI_SlaveTransferCreateHandleEDMA (**LPSPI_Type** * *base*, **lpspi_slave_edma_handle_t** * *handle*, **lpspi_slave_edma_transfer_callback_t** *callback*, **void** * *userData*, **edma_handle_t** * *edmaRxRegToRxDataHandle*, **edma_handle_t** * *edmaTxDataToTxRegHandle*)

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

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	LPSPI handle pointer to <code>lpspi_slave_edma_handle_t</code> .
<i>callback</i>	LPSPI callback.
<i>userData</i>	callback function parameter.
<i>edmaRxRegTo-RxDataHandle</i>	edmaRxRegToRxDataHandle pointer to <code>edma_handle_t</code> .
<i>edmaTxData-ToTxReg-Handle</i>	edmaTxDataToTxRegHandle pointer to <code>edma_handle_t</code> .

24.3.5.8 `status_t LPSPI_SlaveTransferEDMA (LPSPI_Type * base, lpspi_slave_edma_handle_t * handle, lpspi_transfer_t * transfer)`

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	pointer to <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state.
<i>transfer</i>	pointer to <code>lpspi_transfer_t</code> structure.

Returns

status of `status_t`.

24.3.5.9 `void LPSPI_SlaveTransferAbortEDMA (LPSPI_Type * base, lpspi_slave_edma_handle_t * handle)`

This function aborts a transfer which is using eDMA.

Parameters

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

24.3.5.10 status_t LPSPI_SlaveTransferGetCountEDMA (**LPSPI_Type * base,** **lpspi_slave_edma_handle_t * handle, size_t * count**)

This function gets the slave eDMA transfer remaining bytes.

Parameters

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the eDMA transaction.

Returns

status of status_t.

24.4 LPSPI FreeRTOS Driver

24.5 LPSPI CMSIS Driver

This section describes the programming interface of the LPSPI 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>.

24.5.1 Function groups

24.5.1.1 LPSPI CMSIS GetVersion Operation

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

24.5.1.2 LPSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

24.5.1.3 LPSPI 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.

24.5.1.4 LPSPI Transfer Operation

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

24.5.1.5 LPSPI Status Operation

This function group gets the LPSPI transfer status.

24.5.1.6 LPSPI 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.

24.5.2 Typical use case

24.5.2.1 Master Operation

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

/*DSPI master init*/
Driver_SPI0.Initialize(DSPI_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();
```

24.5.2.2 Slave Operation

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

/*DSPI slave init*/
Driver_SPI2.Initialize(DSPI_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 25

LPTMR: Low-Power Timer

25.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

25.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

25.2.1 Initialization and deinitialization

The function [LPTMR_Init\(\)](#) initializes the LPTMR with specified configurations. The function [LPTMR_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the LPTMR for a timer or a pulse counter mode mode. It also sets up the LPTMR's free running mode operation and a clock source.

The function [LPTMR_DeInit\(\)](#) disables the LPTMR module and gates the module clock.

25.2.2 Timer period Operations

The function [LPTMR_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers counts from 0 to the count value set here.

The function [LPTMR_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value ranging from 0 to a timer period.

The timer period operation function takes the count value in ticks. Call the utility macros provided in the `fsl_common.h` file to convert to microseconds or milliseconds.

25.2.3 Start and Stop timer operations

The function [LPTMR_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer counts up to the counter value set earlier by using the [LPTMR_SetPeriod\(\)](#) function. Each time the timer reaches the count value and increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

The function [LPTMR_StopTimer\(\)](#) stops the timer counting and resets the timer's counter register.

25.2.4 Status

Provides functions to get and clear the LPTMR status.

25.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

25.3 Typical use case

25.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lptmr

Data Structures

- struct [lptmr_config_t](#)
LPTMR config structure. [More...](#)

Enumerations

- enum [lptmr_pin_select_t](#) {

 kLPTMR_PinSelectInput_0 = 0x0U,

 kLPTMR_PinSelectInput_1 = 0x1U,

 kLPTMR_PinSelectInput_2 = 0x2U,

 kLPTMR_PinSelectInput_3 = 0x3U }

LPTMR pin selection used in pulse counter mode.
- enum [lptmr_pin_polarity_t](#) {

 kLPTMR_PinPolarityActiveHigh = 0x0U,

 kLPTMR_PinPolarityActiveLow = 0x1U }

LPTMR pin polarity used in pulse counter mode.
- enum [lptmr_timer_mode_t](#) {

 kLPTMR_TimerModeTimeCounter = 0x0U,

 kLPTMR_TimerModePulseCounter = 0x1U }

LPTMR timer mode selection.
- enum [lptmr_prescaler_glitch_value_t](#) {

```

kLPTMR_Prescale_Glitch_0 = 0x0U,
kLPTMR_Prescale_Glitch_1 = 0x1U,
kLPTMR_Prescale_Glitch_2 = 0x2U,
kLPTMR_Prescale_Glitch_3 = 0x3U,
kLPTMR_Prescale_Glitch_4 = 0x4U,
kLPTMR_Prescale_Glitch_5 = 0x5U,
kLPTMR_Prescale_Glitch_6 = 0x6U,
kLPTMR_Prescale_Glitch_7 = 0x7U,
kLPTMR_Prescale_Glitch_8 = 0x8U,
kLPTMR_Prescale_Glitch_9 = 0x9U,
kLPTMR_Prescale_Glitch_10 = 0xAU,
kLPTMR_Prescale_Glitch_11 = 0xBU,
kLPTMR_Prescale_Glitch_12 = 0xCU,
kLPTMR_Prescale_Glitch_13 = 0xDU,
kLPTMR_Prescale_Glitch_14 = 0xEU,
kLPTMR_Prescale_Glitch_15 = 0xFU }

```

LPTMR prescaler/glitch filter values.

- enum `lptmr_prescaler_clock_select_t` {

kLPTMR_PrescalerClock_0 = 0x0U,

kLPTMR_PrescalerClock_2 = 0x2U }
 - enum `lptmr_interrupt_enable_t` { `kLPTMR_TimerInterruptEnable` = LPTMR_CSR_TIE_MASK }
 - enum `lptmr_status_flags_t` { `kLPTMR_TimerCompareFlag` = LPTMR_CSR_TCF_MASK }
- List of the LPTMR interrupts.*
- List of the LPTMR status flags.*

Functions

- static void `LPTMR_EnableTimerDMA` (`LPTMR_Type` *base, bool enable)

Enable or disable timer DMA request.

Driver version

- #define `FSL_LPTMR_DRIVER_VERSION` (`MAKE_VERSION`(2, 2, 0))

Driver Version.

Initialization and deinitialization

- void `LPTMR_Init` (`LPTMR_Type` *base, const `lptmr_config_t` *config)

Ungates the LPTMR clock and configures the peripheral for a basic operation.
- void `LPTMR_Deinit` (`LPTMR_Type` *base)

Gates the LPTMR clock.
- void `LPTMR_GetDefaultConfig` (`lptmr_config_t` *config)

Fills in the LPTMR configuration structure with default settings.

Interrupt Interface

- static void `LPTMR_EnableInterrupts` (`LPTMR_Type` *base, `uint32_t` mask)

- static void [LPTMR_DisableInterrupts](#) (LPTMR_Type *base, uint32_t mask)

Enables the selected LPTMR interrupts.
- static void [LPTMR_DisableInterrupts](#) (LPTMR_Type *base, uint32_t mask)

Disables the selected LPTMR interrupts.
- static uint32_t [LPTMR_GetEnabledInterrupts](#) (LPTMR_Type *base)

Gets the enabled LPTMR interrupts.

Status Interface

- static uint32_t [LPTMR_GetStatusFlags](#) (LPTMR_Type *base)

Gets the LPTMR status flags.
- static void [LPTMR_ClearStatusFlags](#) (LPTMR_Type *base, uint32_t mask)

Clears the LPTMR status flags.

Read and write the timer period

- static void [LPTMR_SetTimerPeriod](#) (LPTMR_Type *base, uint32_t ticks)

Sets the timer period in units of count.
- static uint32_t [LPTMR_GetCurrentTimerCount](#) (LPTMR_Type *base)

Reads the current timer counting value.

Timer Start and Stop

- static void [LPTMR_StartTimer](#) (LPTMR_Type *base)

Starts the timer.
- static void [LPTMR_StopTimer](#) (LPTMR_Type *base)

Stops the timer.

25.4 Data Structure Documentation

25.4.1 struct lptmr_config_t

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the [LPTMR_GetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

Data Fields

- [lptmr_timer_mode_t timerMode](#)

Time counter mode or pulse counter mode.
- [lptmr_pin_select_t pinSelect](#)

LPTMR pulse input pin select; used only in pulse counter mode.
- [lptmr_pin_polarity_t pinPolarity](#)

LPTMR pulse input pin polarity; used only in pulse counter mode.
- bool [enableFreeRunning](#)

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set.
- bool [bypassPrescaler](#)

- `lptmr_prescaler_clock_select_t prescalerClockSource`
LPTMR clock source.
- `lptmr_prescaler_glitch_value_t value`
Prescaler or glitch filter value.

25.5 Enumeration Type Documentation

25.5.1 enum lptmr_pin_select_t

Enumerator

- `kLPTMR_PinSelectInput_0` Pulse counter input 0 is selected.
- `kLPTMR_PinSelectInput_1` Pulse counter input 1 is selected.
- `kLPTMR_PinSelectInput_2` Pulse counter input 2 is selected.
- `kLPTMR_PinSelectInput_3` Pulse counter input 3 is selected.

25.5.2 enum lptmr_pin_polarity_t

Enumerator

- `kLPTMR_PinPolarityActiveHigh` Pulse Counter input source is active-high.
- `kLPTMR_PinPolarityActiveLow` Pulse Counter input source is active-low.

25.5.3 enum lptmr_timer_mode_t

Enumerator

- `kLPTMR_TimerModeTimeCounter` Time Counter mode.
- `kLPTMR_TimerModePulseCounter` Pulse Counter mode.

25.5.4 enum lptmr_prescaler_glitch_value_t

Enumerator

- `kLPTMR_Prescale_Glitch_0` Prescaler divide 2, glitch filter does not support this setting.
- `kLPTMR_Prescale_Glitch_1` Prescaler divide 4, glitch filter 2.
- `kLPTMR_Prescale_Glitch_2` Prescaler divide 8, glitch filter 4.
- `kLPTMR_Prescale_Glitch_3` Prescaler divide 16, glitch filter 8.
- `kLPTMR_Prescale_Glitch_4` Prescaler divide 32, glitch filter 16.
- `kLPTMR_Prescale_Glitch_5` Prescaler divide 64, glitch filter 32.
- `kLPTMR_Prescale_Glitch_6` Prescaler divide 128, glitch filter 64.

- kLPTMR_Prescale_Glitch_7* Prescaler divide 256, glitch filter 128.
- kLPTMR_Prescale_Glitch_8* Prescaler divide 512, glitch filter 256.
- kLPTMR_Prescale_Glitch_9* Prescaler divide 1024, glitch filter 512.
- kLPTMR_Prescale_Glitch_10* Prescaler divide 2048, glitch filter 1024.
- kLPTMR_Prescale_Glitch_11* Prescaler divide 4096, glitch filter 2048.
- kLPTMR_Prescale_Glitch_12* Prescaler divide 8192, glitch filter 4096.
- kLPTMR_Prescale_Glitch_13* Prescaler divide 16384, glitch filter 8192.
- kLPTMR_Prescale_Glitch_14* Prescaler divide 32768, glitch filter 16384.
- kLPTMR_Prescale_Glitch_15* Prescaler divide 65536, glitch filter 32768.

25.5.5 enum lptmr_prescaler_clock_select_t

Note

Clock connections are SoC-specific

Enumerator

- kLPTMR_PrescalerClock_0* Prescaler/glitch filter clock 0 selected.
- kLPTMR_PrescalerClock_2* Prescaler/glitch filter clock 2 selected.

25.5.6 enum lptmr_interrupt_enable_t

Enumerator

- kLPTMR_TimerInterruptEnable* Timer interrupt enable.

25.5.7 enum lptmr_status_flags_t

Enumerator

- kLPTMR_TimerCompareFlag* Timer compare flag.

25.6 Function Documentation

25.6.1 void LPTMR_Init (LPTMR_Type * *base*, const lptmr_config_t * *config*)

Note

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

Parameters

<i>base</i>	LPTMR peripheral base address
<i>config</i>	A pointer to the LPTMR configuration structure.

25.6.2 void LPTMR_Deinit (LPTMR_Type * *base*)

Parameters

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

25.6.3 void LPTMR_GetDefaultConfig (Iptmr_config_t * *config*)

The default values are as follows.

```
*     config->timerMode = kLPTMR_TimerModeTimeCounter;
*     config->pinSelect = kLPTMR_PinSelectInput_0;
*     config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
*     config->enableFreeRunning = false;
*     config->bypassPrescaler = true;
*     config->prescalerClockSource = kLPTMR_PrescalerClock_1;
*     config->value = kLPTMR_Prescale_Glitch_0;
*
```

Parameters

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

25.6.4 static void LPTMR_EnableInterrupts (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	LPTMR peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration Iptmr_interrupt_enable_t

25.6.5 static void LPTMR_DisableInterrupts (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	LPTMR peripheral base address
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration lptmr_interrupt_enable_t .

25.6.6 static uint32_t LPTMR_GetEnabledInterrupts (LPTMR_Type * *base*) [inline], [static]

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [lptmr_interrupt_enable_t](#)

25.6.7 static void LPTMR_EnableTimerDMA (LPTMR_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	base LPTMR peripheral base address
<i>enable</i>	Switcher of timer DMA feature. "true" means to enable, "false" means to disable.

25.6.8 static uint32_t LPTMR_GetStatusFlags (LPTMR_Type * *base*) [inline], [static]

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [lptmr_status_flags_t](#)

25.6.9 **static void LPTMR_ClearStatusFlags (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]**

Parameters

<i>base</i>	LPTMR peripheral base address
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration lptmr_status_flags_t .

25.6.10 static void LPTMR_SetTimerPeriod (LPTMR_Type * *base*, uint32_t *ticks*) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note

1. The TCF flag is set with the CNR equals the count provided here and then increments.
2. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

<i>base</i>	LPTMR peripheral base address
<i>ticks</i>	A timer period in units of ticks, which should be equal or greater than 1.

25.6.11 static uint32_t LPTMR_GetCurrentTimerCount (LPTMR_Type * *base*) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

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

Returns

The current counter value in ticks

**25.6.12 static void LPTMR_StartTimer (LPTMR_Type * *base*) [inline],
[static]**

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

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

**25.6.13 static void LPTMR_StopTimer (LPTMR_Type * *base*) [inline],
[static]**

This function stops the timer and resets the timer's counter register.

Parameters

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

Chapter 26

LPUART: Low Power Universal Asynchronous Receiver-/Transmitter Driver

26.1 Overview

Modules

- [LPUART CMSIS Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)
- [LPUART eDMA Driver](#)

26.2 LPUART Driver

26.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

26.2.2 Typical use case

26.2.2.1 LPUART Operation

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

Data Structures

- struct [lpuart_config_t](#)
LPUART configuration structure. [More...](#)
- struct [lpuart_transfer_t](#)
LPUART transfer structure. [More...](#)
- struct [lpuart_handle_t](#)
LPUART handle structure. [More...](#)

Macros

- #define [UART_RETRY_TIMES](#) 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */
Retry times for waiting flag.

Typedefs

- typedef void(* [lpuart_transfer_callback_t](#))(LPUART_Type *base, lpuart_handle_t *handle, [status_t](#) status, void *userData)
LPUART transfer callback function.

Enumerations

- enum {

kStatus_LPUART_TxBusy = MAKE_STATUS(kStatusGroup_LPUART, 0),

kStatus_LPUART_RxBusy = MAKE_STATUS(kStatusGroup_LPUART, 1),

kStatus_LPUART_TxIdle = MAKE_STATUS(kStatusGroup_LPUART, 2),

kStatus_LPUART_RxIdle = MAKE_STATUS(kStatusGroup_LPUART, 3),

kStatus_LPUART_TxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 4),

kStatus_LPUART_RxWatermarkTooLarge = MAKE_STATUS(kStatusGroup_LPUART, 5),

kStatus_LPUART_FlagCannotClearManually = MAKE_STATUS(kStatusGroup_LPUART, 6),

kStatus_LPUART_Error = MAKE_STATUS(kStatusGroup_LPUART, 7),

kStatus_LPUART_RxRingBufferOverrun,

kStatus_LPUART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_LPUART, 9),

kStatus_LPUART_NoiseError = MAKE_STATUS(kStatusGroup_LPUART, 10),

kStatus_LPUART_FramingError = MAKE_STATUS(kStatusGroup_LPUART, 11),

kStatus_LPUART_ParityError = MAKE_STATUS(kStatusGroup_LPUART, 12),

kStatus_LPUART_BaudrateNotSupport,

kStatus_LPUART_IdleLineDetected = MAKE_STATUS(kStatusGroup_LPUART, 14),

kStatus_LPUART_Timeout = MAKE_STATUS(kStatusGroup_LPUART, 15) }

Error codes for the LPUART driver.

- enum `lpuart_parity_mode_t` {

kLPUART_ParityDisabled = 0x0U,

kLPUART_ParityEven = 0x2U,

kLPUART_ParityOdd = 0x3U }
- LPUART parity mode.*
- enum `lpuart_data_bits_t` {

kLPUART_EightDataBits = 0x0U,

kLPUART_SevenDataBits = 0x1U }
- LPUART data bits count.*
- enum `lpuart_stop_bit_count_t` {

kLPUART_OneStopBit = 0U,

kLPUART_TwoStopBit = 1U }
- LPUART stop bit count.*
- enum `lpuart_transmit_cts_source_t` {

kLPUART_CtsSourcePin = 0U,

kLPUART_CtsSourceMatchResult = 1U }
- LPUART transmit CTS source.*
- enum `lpuart_transmit_cts_config_t` {

kLPUART_CtsSampleAtStart = 0U,

kLPUART_CtsSampleAtIdle = 1U }
- LPUART transmit CTS configure.*
- enum `lpuart_idle_type_select_t` {

kLPUART_IdleTypeStartBit = 0U,

kLPUART_IdleTypeStopBit = 1U }
- LPUART idle flag type defines when the receiver starts counting.*
- enum `lpuart_idle_config_t` {

```
kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
```

LPUART idle detected configuration.

- enum `_lpuart_interrupt_enable` {


```
kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIIE_MASK >> 8U),
kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8U),
kLPUART_TxDataRegEmptyInterruptEnable = (LPUART_CTRL_TIE_MASK),
kLPUART_TransmissionCompleteInterruptEnable = (LPUART_CTRL_TCIE_MASK),
kLPUART_RxDataRegFullInterruptEnable = (LPUART_CTRL_RIE_MASK),
kLPUART_IdleLineInterruptEnable = (LPUART_CTRL_ILIE_MASK),
kLPUART_RxOverrunInterruptEnable = (LPUART_CTRL_ORIE_MASK),
kLPUART_NoiseErrorInterruptEnable = (LPUART_CTRL_NEIE_MASK),
kLPUART_FramingErrorInterruptEnable = (LPUART_CTRL_FEIE_MASK),
kLPUART_ParityErrorInterruptEnable = (LPUART_CTRL_PEIE_MASK),
kLPUART_Match1InterruptEnable = (LPUART_CTRL_MA1IE_MASK),
kLPUART_Match2InterruptEnable = (LPUART_CTRL_MA2IE_MASK),
kLPUART_TxFifoOverflowInterruptEnable = (LPUART_FIFO_TXOFE_MASK),
kLPUART_RxFifoUnderflowInterruptEnable = (LPUART_FIFO_RXUFE_MASK) }
```

LPUART interrupt configuration structure, default settings all disabled.

- enum `_lpuart_flags` {


```
kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag = (LPUART_STAT_RDRF_MASK),
kLPUART_IdleLineFlag = (LPUART_STAT_IDLE_MASK),
kLPUART_RxOverrunFlag = (LPUART_STAT_OR_MASK),
kLPUART_NoiseErrorFlag = (LPUART_STAT_NF_MASK),
kLPUART_FramingErrorFlag,
kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
kLPUART_LinBreakFlag = (LPUART_STAT_LBKDIF_MASK),
kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag,
kLPUART_DataMatch2Flag,
kLPUART_TxFifoEmptyFlag,
kLPUART_RxFifoEmptyFlag,
kLPUART_TxFifoOverflowFlag,
kLPUART_RxFifoUnderflowFlag )
```

LPUART status flags.

Driver version

- #define **FSL_LPUART_DRIVER_VERSION** (MAKE_VERSION(2, 7, 6))
LPUART driver version.

Software Reset

- static void **LPUART_SoftwareReset** (LPUART_Type *base)
Resets the LPUART using software.

Initialization and deinitialization

- **status_t LPUART_Init** (LPUART_Type *base, const **lpuart_config_t** *config, uint32_t srcClock_Hz)
Initializes an LPUART instance with the user configuration structure and the peripheral clock.
- void **LPUART_Deinit** (LPUART_Type *base)
Deinitializes a LPUART instance.
- void **LPUART_GetDefaultConfig** (**lpuart_config_t** *config)
Gets the default configuration structure.

Module configuration

- **status_t LPUART_SetBaudRate** (LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
Sets the LPUART instance baudrate.
- void **LPUART_Enable9bitMode** (LPUART_Type *base, bool enable)
Enable 9-bit data mode for LPUART.
- static void **LPUART_SetMatchAddress** (LPUART_Type *base, uint16_t address1, uint16_t address2)
Set the LPUART address.
- static void **LPUART_EnableMatchAddress** (LPUART_Type *base, bool match1, bool match2)
Enable the LPUART match address feature.
- static void **LPUART_SetRxFifoWatermark** (LPUART_Type *base, uint8_t water)
Sets the rx FIFO watermark.
- static void **LPUART_SetTxFifoWatermark** (LPUART_Type *base, uint8_t water)
Sets the tx FIFO watermark.

Status

- uint32_t **LPUART_GetStatusFlags** (LPUART_Type *base)
Gets LPUART status flags.
- **status_t LPUART_ClearStatusFlags** (LPUART_Type *base, uint32_t mask)
Clears status flags with a provided mask.

Interrupts

- void [LPUART_EnableInterrupts](#) (LPUART_Type *base, uint32_t mask)
Enables LPUART interrupts according to a provided mask.
- void [LPUART_DisableInterrupts](#) (LPUART_Type *base, uint32_t mask)
Disables LPUART interrupts according to a provided mask.
- uint32_t [LPUART_GetEnabledInterrupts](#) (LPUART_Type *base)
Gets enabled LPUART interrupts.

DMA Configuration

- static uintptr_t [LPUART_GetDataRegisterAddress](#) (LPUART_Type *base)
Gets the LPUART data register address.
- static void [LPUART_EnableTxDMA](#) (LPUART_Type *base, bool enable)
Enables or disables the LPUART transmitter DMA request.
- static void [LPUART_EnableRxDMA](#) (LPUART_Type *base, bool enable)
Enables or disables the LPUART receiver DMA.

Bus Operations

- uint32_t [LPUARTGetInstance](#) (LPUART_Type *base)
Get the LPUART instance from peripheral base address.
- static void [LPUART_EnableTx](#) (LPUART_Type *base, bool enable)
Enables or disables the LPUART transmitter.
- static void [LPUART_EnableRx](#) (LPUART_Type *base, bool enable)
Enables or disables the LPUART receiver.
- static void [LPUART_WriteByte](#) (LPUART_Type *base, uint8_t data)
Writes to the transmitter register.
- static uint8_t [LPUART_ReadByte](#) (LPUART_Type *base)
Reads the receiver register.
- static uint8_t [LPUART_GetRxFifoCount](#) (LPUART_Type *base)
Gets the rx FIFO data count.
- static uint8_t [LPUART_GetTxFifoCount](#) (LPUART_Type *base)
Gets the tx FIFO data count.
- void [LPUART_SendAddress](#) (LPUART_Type *base, uint8_t address)
Transmit an address frame in 9-bit data mode.
- status_t [LPUART_WriteBlocking](#) (LPUART_Type *base, const uint8_t *data, size_t length)
Writes to the transmitter register using a blocking method.
- status_t [LPUART_ReadBlocking](#) (LPUART_Type *base, uint8_t *data, size_t length)
Reads the receiver data register using a blocking method.

Transactional

- void [LPUART_TransferCreateHandle](#) (LPUART_Type *base, lpuart_handle_t *handle, [lpuart_transfer_callback_t](#) callback, void *userData)
Initializes the LPUART handle.

- `status_t LPUART_TransferSendNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer)`
Transmits a buffer of data using the interrupt method.
- `void LPUART_TransferStartRingBuffer (LPUART_Type *base, lpuart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)`
Sets up the RX ring buffer.
- `void LPUART_TransferStopRingBuffer (LPUART_Type *base, lpuart_handle_t *handle)`
Aborts the background transfer and uninstalls the ring buffer.
- `size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type *base, lpuart_handle_t *handle)`
Get the length of received data in RX ring buffer.
- `void LPUART_TransferAbortSend (LPUART_Type *base, lpuart_handle_t *handle)`
Aborts the interrupt-driven data transmit.
- `status_t LPUART_TransferGetSendCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`
Gets the number of bytes that have been sent out to bus.
- `status_t LPUART_TransferReceiveNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer, size_t *receivedBytes)`
Receives a buffer of data using the interrupt method.
- `void LPUART_TransferAbortReceive (LPUART_Type *base, lpuart_handle_t *handle)`
Aborts the interrupt-driven data receiving.
- `status_t LPUART_TransferGetReceiveCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`
Gets the number of bytes that have been received.
- `void LPUART_TransferHandleIRQ (LPUART_Type *base, void *irqHandle)`
LPUART IRQ handle function.
- `void LPUART_TransferHandleErrorIRQ (LPUART_Type *base, void *irqHandle)`
LPUART Error IRQ handle function.

26.2.3 Data Structure Documentation

26.2.3.1 struct lpuart_config_t

Data Fields

- `uint32_t baudRate_Bps`
LPUART baud rate.
- `lpuart_parity_mode_t parityMode`
Parity mode, disabled (default), even, odd.
- `lpuart_data_bits_t dataBitsCount`
Data bits count, eight (default), seven.
- `bool isMsb`
Data bits order, LSB (default), MSB.
- `lpuart_stop_bit_count_t stopBitCount`
Number of stop bits, 1 stop bit (default) or 2 stop bits.
- `uint8_t txFifoWatermark`
TX FIFO watermark.
- `uint8_t rxFifoWatermark`

- `bool enableRxRTS`
RX RTS enable.
- `bool enableTxCTS`
TX CTS enable.
- `lpuart_transmit_cts_source_t txCtsSource`
TX CTS source.
- `lpuart_transmit_cts_config_t txCtsConfig`
TX CTS configure.
- `lpuart_idle_type_select_t rxIdleType`
RX IDLE type.
- `lpuart_idle_config_t rxIdleConfig`
RX IDLE configuration.
- `bool enableTx`
Enable TX.
- `bool enableRx`
Enable RX.

Field Documentation

(1) `lpuart_idle_type_select_t lpuart_config_t::rxIdleType`

(2) `lpuart_idle_config_t lpuart_config_t::rxIdleConfig`

26.2.3.2 struct lpuart_transfer_t

Data Fields

- `size_t dataSize`
The byte count to be transfer.
- `uint8_t * data`
The buffer of data to be transfer.
- `uint8_t * rxData`
The buffer to receive data.
- `const uint8_t * txData`
The buffer of data to be sent.

Field Documentation

- (1) `uint8_t* lpuart_transfer_t::data`
- (2) `uint8_t* lpuart_transfer_t::rxData`
- (3) `const uint8_t* lpuart_transfer_t::txData`
- (4) `size_t lpuart_transfer_t::dataSize`

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

Field Documentation

- (1) const uint8_t* volatile lpuart_handle_t::txData
- (2) volatile size_t lpuart_handle_t::txDataSize
- (3) size_t lpuart_handle_t::txDataSizeAll
- (4) uint8_t* volatile lpuart_handle_t::rxData
- (5) volatile size_t lpuart_handle_t::rxDataSize
- (6) size_t lpuart_handle_t::rxDataSizeAll
- (7) uint8_t* lpuart_handle_t::rxRingBuffer
- (8) size_t lpuart_handle_t::rxRingBufferSize
- (9) volatile uint16_t lpuart_handle_t::rxRingBufferHead
- (10) volatile uint16_t lpuart_handle_t::rxRingBufferTail
- (11) lpuart_transfer_callback_t lpuart_handle_t::callback
- (12) void* lpuart_handle_t::userData
- (13) volatile uint8_t lpuart_handle_t::txState
- (14) volatile uint8_t lpuart_handle_t::rxState
- (15) bool lpuart_handle_t::isSevenDataBits

26.2.4 Macro Definition Documentation

26.2.4.1 #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 7, 6))

26.2.4.2 #define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */

26.2.5 Typedef Documentation

26.2.5.1 typedef void(* lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle, status_t status, void *userData)

26.2.6 Enumeration Type Documentation

26.2.6.1 anonymous enum

Enumerator

kStatus_LPUART_RxBusy RX busy.
kStatus_LPUART_TxIdle LPUART transmitter is idle.
kStatus_LPUART_RxIdle LPUART receiver is idle.
kStatus_LPUART_TxWatermarkTooLarge TX FIFO watermark too large.
kStatus_LPUART_RxWatermarkTooLarge RX FIFO watermark too large.
kStatus_LPUART_FlagCannotClearManually Some flag can't manually clear.
kStatus_LPUART_Error Error happens on LPUART.
kStatus_LPUART_RxRingBufferOverrun LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverrun LPUART RX receiver overrun.
kStatus_LPUART_NoiseError LPUART noise error.
kStatus_LPUART_FramingError LPUART framing error.
kStatus_LPUART_ParityError LPUART parity error.
kStatus_LPUART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_LPUART_IdleLineDetected IDLE flag.
kStatus_LPUART_Timeout LPUART times out.

26.2.6.2 enum lpuart_parity_mode_t

Enumerator

kLPUART_ParityDisabled Parity disabled.
kLPUART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kLPUART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

26.2.6.3 enum lpuart_data_bits_t

Enumerator

kLPUART_EightDataBits Eight data bit.
kLPUART_SevenDataBits Seven data bit.

26.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.
kLPUART_TwoStopBit Two stop bits.

26.2.6.5 enum lpuart_transmit_cts_source_t

Enumerator

kLPUART_CtsSourcePin CTS resource is the LPUART_CTS pin.

kLPUART_CtsSourceMatchResult CTS resource is the match result.

26.2.6.6 **enum lpuart_transmit_cts_config_t**

Enumerator

kLPUART_CtsSampleAtStart CTS input is sampled at the start of each character.

kLPUART_CtsSampleAtIdle CTS input is sampled when the transmitter is idle.

26.2.6.7 **enum lpuart_idle_type_select_t**

Enumerator

kLPUART_IdleTypeStartBit Start counting after a valid start bit.

kLPUART_IdleTypeStopBit Start counting after a stop bit.

26.2.6.8 **enum lpuart_idle_config_t**

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

kLPUART_IdleCharacter1 the number of idle characters.

kLPUART_IdleCharacter2 the number of idle characters.

kLPUART_IdleCharacter4 the number of idle characters.

kLPUART_IdleCharacter8 the number of idle characters.

kLPUART_IdleCharacter16 the number of idle characters.

kLPUART_IdleCharacter32 the number of idle characters.

kLPUART_IdleCharacter64 the number of idle characters.

kLPUART_IdleCharacter128 the number of idle characters.

26.2.6.9 **enum _lpuart_interrupt_enable**

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

kLPUART_LinBreakInterruptEnable LIN break detect. bit 7

kLPUART_RxActiveEdgeInterruptEnable Receive Active Edge. bit 6

kLPUART_TxDataRegEmptyInterruptEnable Transmit data register empty. bit 23

kLPUART_TransmissionCompleteInterruptEnable Transmission complete. bit 22

kLPUART_RxDataRegFullInterruptEnable Receiver data register full. bit 21

kLPUART_IdleLineInterruptEnable Idle line. bit 20
kLPUART_RxOverrunInterruptEnable Receiver Overrun. bit 27
kLPUART_NoiseErrorInterruptEnable Noise error flag. bit 26
kLPUART_FramingErrorInterruptEnable Framing error flag. bit 25
kLPUART_ParityErrorInterruptEnable Parity error flag. bit 24
kLPUART_Match1InterruptEnable Parity error flag. bit 15
kLPUART_Match2InterruptEnable Parity error flag. bit 14
kLPUART_TxFifoOverflowInterruptEnable Transmit FIFO Overflow. bit 9
kLPUART_RxFifoUnderflowInterruptEnable Receive FIFO Underflow. bit 8

26.2.6.10 enum _lpuart_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

kLPUART_TxDataRegEmptyFlag Transmit data register empty flag, sets when transmit buffer is empty. bit 23
kLPUART_TransmissionCompleteFlag Transmission complete flag, sets when transmission activity complete. bit 22
kLPUART_RxDataRegFullFlag Receive data register full flag, sets when the receive data buffer is full. bit 21
kLPUART_IdleLineFlag Idle line detect flag, sets when idle line detected. bit 20
kLPUART_RxOverrunFlag Receive Overrun, sets when new data is received before data is read from receive register. bit 19
kLPUART_NoiseErrorFlag Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18
kLPUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17
kLPUART_ParityErrorFlag If parity enabled, sets upon parity error detection. bit 16
kLPUART_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31
kLPUART_RxActiveEdgeFlag Receive pin active edge interrupt flag, sets when active edge detected. bit 30
kLPUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start. bit 24
kLPUART_DataMatch1Flag The next character to be read from LPUART_DATA matches MA1. bit 15
kLPUART_DataMatch2Flag The next character to be read from LPUART_DATA matches MA2. bit 14
kLPUART_TxFifoEmptyFlag TXEMPT bit, sets if transmit buffer is empty. bit 7
kLPUART_RxFifoEmptyFlag RXEMPT bit, sets if receive buffer is empty. bit 6
kLPUART_TxFifoOverflowFlag TXOF bit, sets if transmit buffer overflow occurred. bit 1
kLPUART_RxFifoUnderflowFlag RXUF bit, sets if receive buffer underflow occurred. bit 0

26.2.7 Function Documentation

26.2.7.1 static void LPUART_SoftwareReset (LPUART_Type * *base*) [inline], [static]

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

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

26.2.7.2 status_t LPUART_Init (LPUART_Type * *base*, const lpuart_config_t * *config*, uint32_t *srcClock_Hz*)

This function configures the LPUART module with user-defined settings. Call the [LPUART_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

Parameters

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

Return values

<i>kStatus_LPUART_BaudrateNotSupport</i>	Baudrate is not support in current clock source.
<i>kStatus_Success</i>	LPUART initialize succeed

26.2.7.3 void LPUART_Deinit (LPUART_Type * *base*)

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

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

26.2.7.4 void LPUART_GetDefaultConfig (*Ipuart_config_t * config*)

This function initializes the LPUART configuration structure to a default value. The default values are:
: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled;
lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

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

26.2.7.5 status_t LPUART_SetBaudRate (*LPUART_Type * base, uint32_t baudRate_Bps, uint32_t srcClock_Hz*)

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

<i>base</i>	LPUART peripheral base address.
<i>baudRate_Bps</i>	LPUART baudrate to be set.
<i>srcClock_Hz</i>	LPUART clock source frequency in HZ.

Return values

<i>kStatus_LPUART_BaudrateNotSupport</i>	Baudrate is not supported in the current clock source.
<i>kStatus_Success</i>	Set baudrate succeeded.

26.2.7.6 void LPUART_Enable9bitMode (*LPUART_Type * base, bool enable*)

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	true to enable, false to disable.

26.2.7.7 static void LPUART_SetMatchAddress (LPUART_Type * *base*, uint16_t *address1*, uint16_t *address2*) [inline], [static]

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>address1</i>	LPUART slave address1.
<i>address2</i>	LPUART slave address2.

26.2.7.8 static void LPUART_EnableMatchAddress (LPUART_Type * *base*, bool *match1*, bool *match2*) [inline], [static]

Parameters

<i>base</i>	LPUART peripheral base address.
<i>match1</i>	true to enable match address1, false to disable.
<i>match2</i>	true to enable match address2, false to disable.

26.2.7.9 static void LPUART_SetRxFifoWatermark (LPUART_Type * *base*, uint8_t *water*) [inline], [static]

Parameters

<i>base</i>	LPUART peripheral base address.
<i>water</i>	Rx FIFO watermark.

26.2.7.10 static void LPUART_SetTxFifoWatermark (LPUART_Type * *base*, uint8_t *water*) [inline], [static]

Parameters

<i>base</i>	LPUART peripheral base address.
<i>water</i>	Tx FIFO watermark.

26.2.7.11 uint32_t LPUART_GetStatusFlags (LPUART_Type * *base*)

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators `_lpuart_flags`. To check for a specific status, compare the return value with enumerators in the `_lpuart_flags`. For example, to check whether the TX is empty:

```
*     if (kLPUART_TxDataRegEmptyFlag &
*         LPUART_GetStatusFlags(LPUART1))
*     {
*     ...
*     }
```

Parameters

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

Returns

LPUART status flags which are ORed by the enumerators in the `_lpuart_flags`.

26.2.7.12 status_t LPUART_ClearStatusFlags (LPUART_Type * *base*, uint32_t *mask*)

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag, kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	the status flags to be cleared. The user can use the enumerators in the <code>_lpuart_status_flag_t</code> to do the OR operation and get the mask.

Returns

0 succeed, others failed.

Return values

<i>kStatus_LPUART_Flag_CannotClearManually</i>	The flag can't be cleared by this function but it is cleared automatically by hardware.
<i>kStatus_Success</i>	Status in the mask are cleared.

26.2.7.13 void LPUART_EnableInterrupts (LPUART_Type * *base*, uint32_t *mask*)

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the [_lpuart_interrupt_enable](#). This examples shows how to enable TX empty interrupt and RX full interrupt:

```
*     LPUART_EnableInterrupts(LPUART1,
    kLPUART_TxDataRegEmptyInterruptEnable |
    kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	The interrupts to enable. Logical OR of _lpuart_interrupt_enable .

26.2.7.14 void LPUART_DisableInterrupts (LPUART_Type * *base*, uint32_t *mask*)

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [_lpuart_interrupt_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     LPUART_DisableInterrupts(LPUART1,
    kLPUART_TxDataRegEmptyInterruptEnable |
    kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

<i>base</i>	LPUART peripheral base address.
<i>mask</i>	The interrupts to disable. Logical OR of _lpuart_interrupt_enable .

26.2.7.15 uint32_t LPUART_GetEnabledInterrupts (LPUART_Type * *base*)

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [_lpuart_interrupt_enable](#). To check a specific interrupt enable status, compare the return value with enumerators in [_lpuart_interrupt_enable](#). For example, to check whether the TX empty interrupt is enabled:

```
*     uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
*     if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
*     {
*         ...
*     }
```

Parameters

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

Returns

LPUART interrupt flags which are logical OR of the enumerators in [_lpuart_interrupt_enable](#).

26.2.7.16 static uintptr_t LPUART_GetDataRegisterAddress (LPUART_Type * *base*) [inline], [static]

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

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

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

26.2.7.17 static void LPUART_EnableTxDMA (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

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

26.2.7.18 static void LPUART_EnableRxDMA (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

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

26.2.7.19 uint32_t LPUART_GetInstance (LPUART_Type * *base*)

Parameters

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

Returns

LPUART instance.

26.2.7.20 static void LPUART_EnableTx (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART transmitter.

Parameters

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

26.2.7.21 static void LPUART_EnableRx (LPUART_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

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

26.2.7.22 static void LPUART_WriteByte (LPUART_Type * *base*, uint8_t *data*) [inline], [static]

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Data write to the TX register.

26.2.7.23 static uint8_t LPUART_ReadByte (LPUART_Type * *base*) [inline], [static]

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

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

Returns

Data read from data register.

26.2.7.24 static uint8_t LPUART_GetRx_fifoCount (LPUART_Type * *base*) [inline], [static]

Parameters

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

Returns

rx FIFO data count.

26.2.7.25 static uint8_t LPUART_GetTxFifoCount (LPUART_Type * *base*) [inline], [static]

Parameters

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

Returns

tx FIFO data count.

26.2.7.26 void LPUART_SendAddress (LPUART_Type * *base*, uint8_t *address*)

Parameters

<i>base</i>	LPUART peripheral base address.
<i>address</i>	LPUART slave address.

26.2.7.27 status_t LPUART_WriteBlocking (LPUART_Type * *base*, const uint8_t * *data*, size_t *length*)

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Start address of the data to write.
<i>length</i>	Size of the data to write.

Return values

<i>kStatus_LPUART_- Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully wrote all data.

26.2.7.28 **status_t LPUART_ReadBlocking (LPUART_Type * *base*, uint8_t * *data*, size_t *length*)**

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Start address of the buffer to store the received data.
<i>length</i>	Size of the buffer.

Return values

<i>kStatus_LPUART_Rx- HardwareOverrun</i>	Receiver overrun happened while receiving data.
<i>kStatus_LPUART_Noise- Error</i>	Noise error happened while receiving data.
<i>kStatus_LPUART_- FramingError</i>	Framing error happened while receiving data.
<i>kStatus_LPUART_Parity- Error</i>	Parity error happened while receiving data.
<i>kStatus_LPUART_- Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully received all data.

26.2.7.29 **void LPUART_TransferCreateHandle (LPUART_Type * *base*, Ipuart_handle_t * *handle*, Ipuart_transfer_callback_t *callback*, void * *userData*)**

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

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user

can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

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

26.2.7.30 `status_t LPUART_TransferSendNonBlocking (LPUART_Type * base, Ipuart_handle_t * handle, Ipuart_transfer_t * xfer)`

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the `kStatus_LPUART_TxIdle` as status parameter.

Note

The `kStatus_LPUART_TxIdle` is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the `kLPUART_TransmissionCompleteFlag` to ensure that the transmit is finished.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>xfer</i>	LPUART transfer structure, see Ipuart_transfer_t .

Return values

<code>kStatus_Success</code>	Successfully start the data transmission.
<code>kStatus_LPUART_TxBusy</code>	Previous transmission still not finished, data not all written to the TX register.
<code>kStatus_InvalidArgument</code>	Invalid argument.

26.2.7.31 `void LPUART_TransferStartRingBuffer (LPUART_Type * base, Ipuart_handle_t * handle, uint8_t * ringBuffer, size_t ringBufferSize)`

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the `UART_TransferReceiveNonBlocking()` API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>ringBuffer</i>	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
<i>ringBufferSize</i>	size of the ring buffer.

26.2.7.32 void LPUART_TransferStopRingBuffer (`LPUART_Type * base, Ipuart_handle_t * handle`)

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

Parameters

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

26.2.7.33 size_t LPUART_TransferGetRxRingBufferLength (`LPUART_Type * base, Ipuart_handle_t * handle`)

Parameters

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

Returns

Length of received data in RX ring buffer.

26.2.7.34 void LPUART_TransferAbortSend (LPUART_Type * *base*, Ipuart_handle_t * *handle*)

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

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

26.2.7.35 status_t LPUART_TransferGetSendCount (**LPUART_Type * base,** **Ipuart_handle_t * handle, uint32_t * count**)

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Send bytes count.

Return values

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

26.2.7.36 status_t LPUART_TransferReceiveNonBlocking (**LPUART_Type * base,** **Ipuart_handle_t * handle, Ipuart_transfer_t * xfer, size_t * receivedBytes**)

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter *kStatus_UART_RxIdle*. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to *xfer->data*, which returns with the parameter *receivedBytes* set to 5. For the remaining 5 bytes, the newly arrived data is saved from *xfer->data[5]*. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to *xfer->data*. When all data is received, the upper layer is notified.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>xfer</i>	LPUART transfer structure, see <code>uart_transfer_t</code> .
<i>receivedBytes</i>	Bytes received from the ring buffer directly.

Return values

<i>kStatus_Success</i>	Successfully queue the transfer into the transmit queue.
<i>kStatus_LPUART_Rx-Busy</i>	Previous receive request is not finished.
<i>kStatus_InvalidArgument</i>	Invalid argument.

26.2.7.37 void LPUART_TransferAbortReceive (`LPUART_Type * base, Ipuart_handle_t * handle`)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

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

26.2.7.38 status_t LPUART_TransferGetReceiveCount (`LPUART_Type * base, Ipuart_handle_t * handle, uint32_t * count`)

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

Parameters

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

Return values

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

26.2.7.39 void LPUART_TransferHandleIRQ (LPUART_Type * *base*, void * *irqHandle*)

This function handles the LPUART transmit and receive IRQ request.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>irqHandle</i>	LPUART handle pointer.

26.2.7.40 void LPUART_TransferHandleErrorIRQ (LPUART_Type * *base*, void * *irqHandle*)

This function handles the LPUART error IRQ request.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>irqHandle</i>	LPUART handle pointer.

26.3 LPUART eDMA Driver

26.3.1 Overview

Data Structures

- struct [lpuart_edma_handle_t](#)
LPUART eDMA handle. [More...](#)

TypeDefs

- [typedef void\(* lpuart_edma_transfer_callback_t \)](#)(LPUART_Type *base, lpuart_edma_handle_t *handle, [status_t](#) status, void *userData)
LPUART transfer callback function.

Driver version

- #define [FSL_LPUART_EDMA_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 6, 0))
LPUART EDMA driver version.

eDMA transactional

- void [LPUART_TransferCreateHandleEDMA](#) (LPUART_Type *base, lpuart_edma_handle_t *handle, [lpuart_edma_transfer_callback_t](#) callback, void *userData, [edma_handle_t](#) *txEdmaHandle, [edma_handle_t](#) *rxEdmaHandle)
Initializes the LPUART handle which is used in transactional functions.
- [status_t LPUART_SendEDMA](#) (LPUART_Type *base, lpuart_edma_handle_t *handle, [lpuart_transfer_t](#) *xfer)
Sends data using eDMA.
- [status_t LPUART_ReceiveEDMA](#) (LPUART_Type *base, lpuart_edma_handle_t *handle, [lpuart_transfer_t](#) *xfer)
Receives data using eDMA.
- void [LPUART_TransferAbortSendEDMA](#) (LPUART_Type *base, lpuart_edma_handle_t *handle)
Aborts the sent data using eDMA.
- void [LPUART_TransferAbortReceiveEDMA](#) (LPUART_Type *base, lpuart_edma_handle_t *handle)
Aborts the received data using eDMA.
- [status_t LPUART_TransferGetSendCountEDMA](#) (LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)
Gets the number of bytes written to the LPUART TX register.
- [status_t LPUART_TransferGetReceiveCountEDMA](#) (LPUART_Type *base, lpuart_edma_handle_t *handle, uint32_t *count)
Gets the number of received bytes.
- void [LPUART_TransferEdmaHandleIRQ](#) (LPUART_Type *base, void *lpuartEdmaHandle)
LPUART eDMA IRQ handle function.

26.3.2 Data Structure Documentation

26.3.2.1 struct _lpuart_edma_handle

Data Fields

- `lpuart_edma_transfer_callback_t callback`
Callback function.
- `void *userData`
LPUART callback function parameter.
- `size_t rxDataSizeAll`
Size of the data to receive.
- `size_t txDataSizeAll`
Size of the data to send out.
- `edma_handle_t *txEdmaHandle`
The eDMA TX channel used.
- `edma_handle_t *rxEdmaHandle`
The eDMA RX channel used.
- `uint8_t nbytes`
eDMA minor byte transfer count initially configured.
- `volatile uint8_t txState`
TX transfer state.
- `volatile uint8_t rxState`
RX transfer state.

Field Documentation

- (1) `lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback`
- (2) `void* lpuart_edma_handle_t::userData`
- (3) `size_t lpuart_edma_handle_t::rxDataSizeAll`
- (4) `size_t lpuart_edma_handle_t::txDataSizeAll`
- (5) `edma_handle_t* lpuart_edma_handle_t::txEdmaHandle`
- (6) `edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle`
- (7) `uint8_t lpuart_edma_handle_t::nbytes`
- (8) `volatile uint8_t lpuart_edma_handle_t::txState`

26.3.3 Macro Definition Documentation

26.3.3.1 #define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

26.3.4 Typedef Documentation

**26.3.4.1 typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base,
lpuart_edma_handle_t *handle, status_t status, void *userData)**

26.3.5 Function Documentation

**26.3.5.1 void LPUART_TransferCreateHandleEDMA (LPUART_Type * *base*,
lpuart_edma_handle_t * *handle*, lpuart_edma_transfer_callback_t *callback*, void
* *userData*, edma_handle_t * *txEdmaHandle*, edma_handle_t * *rxEdmaHandle*)**

Note

This function disables all LPUART interrupts.

Parameters

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

<i>handle</i>	Pointer to lpuart_edma_handle_t structure.
<i>callback</i>	Callback function.
<i>userData</i>	User data.
<i>txEdmaHandle</i>	User requested DMA handle for TX DMA transfer.
<i>rxEdmaHandle</i>	User requested DMA handle for RX DMA transfer.

26.3.5.2 status_t LPUART_SendEDMA (LPUART_Type * *base*, Ipuart_edma_handle_t * *handle*, Ipuart_transfer_t * *xfer*)

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>xfer</i>	LPUART eDMA transfer structure. See Ipuart_transfer_t .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_LPUART_TxBusy</i>	Previous transfer on going.
<i>kStatus_InvalidArgument</i>	Invalid argument.

26.3.5.3 status_t LPUART_ReceiveEDMA (LPUART_Type * *base*, Ipuart_edma_handle_t * *handle*, Ipuart_transfer_t * *xfer*)

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_edma_handle_t structure.

<i>xfer</i>	LPUART eDMA transfer structure, see lpuart_transfer_t .
-------------	---

Return values

<i>kStatus_Success</i>	if succeed, others fail.
<i>kStatus_LPUART_Rx-Busy</i>	Previous transfer ongoing.
<i>kStatus_InvalidArgument</i>	Invalid argument.

26.3.5.4 void LPUART_TransferAbortSendEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*)

This function aborts the sent data using eDMA.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_edma_handle_t structure.

26.3.5.5 void LPUART_TransferAbortReceiveEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*)

This function aborts the received data using eDMA.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	Pointer to lpuart_edma_handle_t structure.

26.3.5.6 status_t LPUART_TransferGetSendCountEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>count</i>	Send bytes count.

Return values

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

26.3.5.7 **status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type * *base*, Ipuart_edma_handle_t * *handle*, uint32_t * *count*)**

This function gets the number of received bytes.

Parameters

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

Return values

<i>kStatus_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> ;

26.3.5.8 **void LPUART_TransferEdmaHandleIRQ (LPUART_Type * *base*, void * *IpuartEdmaHandle*)**

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note

This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>lpuartEdma-Handle</i>	LPUART handle pointer.

26.4 LPUART FreeRTOS Driver

26.4.1 Overview

Data Structures

- struct `lpuart_rtos_config_t`
LPUART RTOS configuration structure. [More...](#)

Driver version

- #define `FSL_LPUART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)
LPUART FreeRTOS driver version.

LPUART RTOS Operation

- int `LPUART_RTOS_Init` (`lpuart_rtos_handle_t *handle, lpuart_handle_t *t_handle, const lpuart_rtos_config_t *cfg`)
Initializes an LPUART instance for operation in RTOS.
- int `LPUART_RTOS_Deinit` (`lpuart_rtos_handle_t *handle`)
Deinitializes an LPUART instance for operation.

LPUART transactional Operation

- int `LPUART_RTOS_Send` (`lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length`)
Sends data in the background.
- int `LPUART_RTOS_Receive` (`lpuart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received`)
Receives data.
- int `LPUART_RTOS_SetRxTimeout` (`lpuart_rtos_handle_t *handle, uint32_t rx_timeout_constant_ms, uint32_t rx_timeout_multiplier_ms`)
Set RX timeout in runtime.
- int `LPUART_RTOS_SetTxTimeout` (`lpuart_rtos_handle_t *handle, uint32_t tx_timeout_constant_ms, uint32_t tx_timeout_multiplier_ms`)
Set TX timeout in runtime.

26.4.2 Data Structure Documentation

26.4.2.1 struct `lpuart_rtos_config_t`

Data Fields

- `LPUART_Type * base`
UART base address.

- `uint32_t srclk`
UART source clock in Hz.
- `uint32_t baudrate`
Desired communication speed.
- `lpuart_parity_mode_t parity`
Parity setting.
- `lpuart_stop_bit_count_t stopbits`
Number of stop bits to use.
- `uint8_t * buffer`
Buffer for background reception.
- `uint32_t buffer_size`
Size of buffer for background reception.
- `uint32_t rx_timeout_constant_ms`
RX timeout applied per receive.
- `uint32_t rx_timeout_multiplier_ms`
RX timeout added for each byte of the receive.
- `uint32_t tx_timeout_constant_ms`
TX timeout applied per transmission.
- `uint32_t tx_timeout_multiplier_ms`
TX timeout added for each byte of the transmission.
- `bool enableRxRTS`
RX RTS enable.
- `bool enableTxCTS`
TX CTS enable.
- `lpuart_transmit_cts_source_t txCtsSource`
TX CTS source.
- `lpuart_transmit_cts_config_t txCtsConfig`
TX CTS configure.

Field Documentation

- (1) `uint32_t lpuart_rtos_config_t::rx_timeout_multiplier_ms`
- (2) `uint32_t lpuart_rtos_config_t::tx_timeout_multiplier_ms`

26.4.3 Macro Definition Documentation

26.4.3.1 #define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

26.4.4 Function Documentation

26.4.4.1 int LPUART_RTOS_Init (`lpuart_rtos_handle_t * handle`, `lpuart_handle_t * t_handle`, `const lpuart_rtos_config_t * cfg`)

Parameters

<i>handle</i>	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
<i>t_handle</i>	The pointer to an allocated space to store the transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

26.4.4.2 int LPUART_RTOS_Deinit (*Ipuart_rtos_handle_t * handle*)

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

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

26.4.4.3 int LPUART_RTOS_Send (*Ipuart_rtos_handle_t * handle, uint8_t * buffer, uint32_t length*)

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

<i>handle</i>	The RTOS LPUART handle.
<i>buffer</i>	The pointer to buffer to send.
<i>length</i>	The number of bytes to send.

26.4.4.4 int LPUART_RTOS_Receive (*Ipuart_rtos_handle_t * handle, uint8_t * buffer, uint32_t length, size_t * received*)

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

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

26.4.4.5 int LPUART_RTOSETXTIMEOUT (Ipuart_rtos_handle_t * *handle*, uint32_t *rx_timeout_constant_ms*, uint32_t *rx_timeout_multiplier_ms*)

This function can modify RX timeout between initialization and receive.

param handle The RTOS LPUART handle. param rx_timeout_constant_ms RX timeout applied per receive. param rx_timeout_multiplier_ms RX timeout added for each byte of the receive.

26.4.4.6 int LPUART_RTOSETXTIMEOUT (Ipuart_rtos_handle_t * *handle*, uint32_t *tx_timeout_constant_ms*, uint32_t *tx_timeout_multiplier_ms*)

This function can modify TX timeout between initialization and send.

param handle The RTOS LPUART handle. param tx_timeout_constant_ms TX timeout applied per transmission. param tx_timeout_multiplier_ms TX timeout added for each byte of the transmission.

26.5 LPUART CMSIS Driver

This section describes the programming interface of the LPUART 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 LPUART 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.

26.5.1 Function groups

26.5.1.1 LPUART CMSIS GetVersion Operation

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

26.5.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

26.5.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialized the lpuart instance . And this API must be called before you configure a lpuart instance or after you Deinit a lpuart 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.

26.5.1.4 LPUART CMSIS Transfer Operation

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

26.5.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

26.5.1.6 LPUART CMSIS Control Operation

This function can configure an instance ,set baudrate for lpuart, get current baudrate ,set transfer data bits and other control command.

Chapter 27

MSGINTR: Message Unit

27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the message unit module of MCUXpresso SDK devices.

The MSGINTR supports to let other modules to trigger interrupt through writing data into MSGINTR register address. The MSGINTR maps different message interrupts to specified bit, one MSGINTR supports 3*32 bit map.

27.2 Typical use case

Modules

- [MSGINTR Driver](#)

27.3 MSGINTR Driver

Chapter 28

MU: Messaging Unit

28.1 Overview

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

28.2 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.

28.2.1 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.

28.2.2 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.

28.2.3 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.

28.2.4 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.

28.2.5 MU misc functions

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

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

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

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

Enumerations

- enum `_mu_status_flags` {

 `kMU_Tx0EmptyFlag` = (1U << (MU_SR_TEEn_SHIFT + 3U)),

 `kMU_Tx1EmptyFlag` = (1U << (MU_SR_TEEn_SHIFT + 2U)),

 `kMU_Tx2EmptyFlag` = (1U << (MU_SR_TEEn_SHIFT + 1U)),

 `kMU_Tx3EmptyFlag` = (1U << (MU_SR_TEEn_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_MuResetInterruptFlag` = MU_SR_MURIP_MASK }

MU status flags.

- enum `_mu_interrupt_enable` {

 `kMU_Tx0EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 3U)),

 `kMU_Tx1EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 2U)),

 `kMU_Tx2EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 1U)),

 `kMU_Tx3EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 0U)),

 `kMU_Rx0FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 3U)),

 `kMU_Rx1FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 2U)),

 `kMU_Rx2FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 1U)),

 `kMU_Rx3FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 0U)),

 `kMU_GenInt0InterruptEnable` = (int)(1U << (MU_CR_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)),

 `kMU_MuResetInterruptEnable` = MU_CR_MURIE_MASK }

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_t`

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

- void [MU_BootCoreB](#) (MU_Type *base, mu_core_boot_mode_t mode)
Boots the core at B side.
- static void [MU_HoldCoreBReset](#) (MU_Type *base)
Holds the core reset of B side.
- void [MU_BootOtherCore](#) (MU_Type *base, mu_core_boot_mode_t mode)
Boots the other core.
- static void [MU_HoldOtherCoreReset](#) (MU_Type *base)

- static void [MU_ResetBothSides](#) (MU_Type *base)

Holds the other core reset.
- static void [MU_HardwareResetOtherCore](#) (MU_Type *base, bool waitReset, bool holdReset, mu_core_boot_mode_t bootMode)

Resets the MU for both A side and B side.
- static mu_power_mode_t [MU_GetOtherCorePowerMode](#) (MU_Type *base)

Hardware reset the other core.
- static mu_power_mode_t [MU_GetOtherCorePowerMode](#) (MU_Type *base)

Gets the power mode of the other core.

28.3 Macro Definition Documentation

28.3.1 #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

28.4 Enumeration Type Documentation

28.4.1 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_MuResetInterruptFlag* The other side initializes MU reset.

28.4.2 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.
kMU_MuResetInterruptEnable The other side initializes MU reset. The interrupt is ORed with the general purpose interrupt 3. The general purpose interrupt 3 is issued when the other side set the MU reset and this interrupt is enabled.

28.4.3 enum _mu_interrupt_trigger

Enumerator

kMU_GenInt0InterruptTrigger General purpose interrupt 0.
kMU_GenInt1InterruptTrigger General purpose interrupt 1.
kMU_GenInt2InterruptTrigger General purpose interrupt 2.
kMU_GenInt3InterruptTrigger General purpose interrupt 3.

28.5 Function Documentation

28.5.1 void MU_Init (MU_Type * *base*)

This function enables the MU clock only.

Parameters

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

28.5.2 void MU_Deinit (MU_Type * *base*)

This function disables the MU clock only.

Parameters

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

28.5.3 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 <code>_mu_msg_reg_index</code> .
<i>msg</i>	Message to send.

28.5.4 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 <code>_mu_msg_reg_index</code> .
<i>msg</i>	Message to send.

28.5.5 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 <code>_mu_msg_reg_index</code> .

Returns

The received message.

28.5.6 `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 <code>_mu_msg_reg_index</code>

Returns

The received message.

28.5.7 `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.

28.5.8 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.

28.5.9 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.

28.5.10 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`.

28.5.11 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.

28.5.12 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"> • <code>kMU_Tx0EmptyFlag</code> • <code>kMU_Tx1EmptyFlag</code> • <code>kMU_Tx2EmptyFlag</code> • <code>kMU_Tx3EmptyFlag</code> • <code>kMU_Rx0FullFlag</code> • <code>kMU_Rx1FullFlag</code> • <code>kMU_Rx2FullFlag</code> • <code>kMU_Rx3FullFlag</code> • <code>kMU_EventPendingFlag</code> • <code>kMU_FlagsUpdatingFlag</code> • <code>kMU_OtherSideInResetFlag</code>

28.5.13 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> .

28.5.14 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> .

28.5.15 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.

28.5.16 void MU_BootCoreB (MU_Type * *base*, mu_core_boot_mode_t *mode*)

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

Parameters

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

<i>mode</i>	Core B boot mode.
-------------	-------------------

Note

Only MU side A can use this function.

28.5.17 static void MU_HoldCoreBReset (MU_Type * *base*) [inline], [static]

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

Parameters

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

Note

Only A side could call this function.

28.5.18 void MU_BootOtherCore (MU_Type * *base*, mu_core_boot_mode_t *mode*)

This function boots the other core with a boot configuration.

Parameters

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

28.5.19 static void MU_HoldOtherCoreReset (MU_Type * *base*) [inline], [static]

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

Parameters

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

28.5.20 static void MU_ResetBothSides (MU_Type * *base*) [inline], [static]

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

Parameters

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

Note

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

28.5.21 void MU_HardwareResetOtherCore (MU_Type * *base*, bool *waitReset*, bool *holdReset*, mu_core_boot_mode_t *bootMode*)

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

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

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

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

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

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

Parameters

<i>base</i>	MU peripheral base address.
<i>waitReset</i>	<p>Wait the other core enters reset.</p> <ul style="list-style-type: none"> • true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked. • false: Don't wait the reset.
<i>holdReset</i>	<p>Hold the other core reset or not.</p> <ul style="list-style-type: none"> • true: Hold the other core in reset, this function returns directly when the other core enters reset. • false: Don't hold the other core in reset, this function waits until the other core out of reset.
<i>bootMode</i>	Boot mode of the other core, if <i>holdReset</i> is true, this parameter is useless.

28.5.22 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 29

NETC driver

29.1 Overview

The MCUXpresso SDK provides a driver for the NETC module. The NETC driver is designed in 2 layers:

- API layer
- Hardware layer

Modules

- [API layer](#)
- [Abbreviation in NETC driver](#)
- [Hardware layer](#)

Data Structures

- struct [netc_vlan_t](#)
VLAN tag struct. [More...](#)
- struct [netc_buffer_struct_t](#)
Buffer structure. [More...](#)
- struct [netc_frame_struct_t](#)
Frame structure for single Tx/Rx frame. [More...](#)
- struct [netc_frame_attr_t](#)
Frame attribute struct. [More...](#)
- struct [netc_tx_frame_info_t](#)
Frame attribute structure. [More...](#)
- struct [netc_msix_entry_t](#)
NETC MSIX entry structure. [More...](#)

Macros

- #define [FSL_NETC_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 6, 1))
Driver Version.
- #define [NETC_ADDR_LOW_32BIT](#)(x) (([uint32_t](#))(x)&0xFFFFFFFFU)
Macro to divides an address into a low 32 bits and a possible high 32 bits.

Enumerations

- enum {

kStatus_NETC_RxFrameEmpty = MAKE_STATUS(kStatusGroup_NETC, 0),

kStatus_NETC_RxTsrResp = MAKE_STATUS(kStatusGroup_NETC, 1),

kStatus_NETC_RxFrameError = MAKE_STATUS(kStatusGroup_NETC, 2),

kStatus_NETC_TxFrameOverLen = MAKE_STATUS(kStatusGroup_NETC, 3),

kStatus_NETC_LackOfResource,

kStatus_NETC_Unsupported = MAKE_STATUS(kStatusGroup_NETC, 5),

kStatus_NETC_RxHRZeroFrame = MAKE_STATUS(kStatusGroup_NETC, 6),

kStatus_NETC_RxHRNotZeroFrame = MAKE_STATUS(kStatusGroup_NETC, 7),

kStatus_NETC_NotFound = MAKE_STATUS(kStatusGroup_NETC, 8),

kStatus_NETC_EntryExists = MAKE_STATUS(kStatusGroup_NETC, 9) }

Status code for the NETC module.
- enum netc_ep_event_t {

kNETC_EPRxEvent,

kNETC_EPTxEvent }

Defines the common interrupt event for callback use.
- enum netc_ep_tx_status_t {

kNETC_EPTxSuccess = 0x0,

kNETC_EPTxProgramErr = 0x1,

kNETC_EPTxTsdDrop = 0x2,

kNETC_EPTxFrameSizeErr = 0x8,

kNETC_EPTxNullAddr = 0x9,

kNETC_EPTxInvalidLength = 0xA,

kNETC_EPTxSrcMacSpoofingDetect = 0x10,

kNETC_EPTxPortRestDrop = 0x20,

kNETC_EPTxPortDisableDrop = 0x21,

kNETC_EPTxVlanTpIdDrop = 0x40,

kNETC_EPTxSmsParamErr = 0x60,

kNETC_EPTxFrameGateErr = 0x80,

kNETC_EPTxAxiReadErr = 0x90,

kNETC_EPTxAxiWriteErr = 0x91,

kNETC_EPTxMultiBitECCErr = 0xA0,

kNETC_EPTxParityErr = 0xF0,

kNETC_EPTxSwCongestion = 0x100 } }

Status for the transmit buffer descriptor status.
- enum netc_vlan_tp_id_select_t {

kNETC_StanCvlan = 0x0U,

kNETC_StanSvlan,

kNETC_CustomVlan1,

kNETC_CustomVlan2 } }

Ethernet VLAN Tag protocol identifier.
- enum netc_packet_type_t

Ethernet packet type enumerator.
- enum netc_host_reason_t

Host reason.

- enum `netc_msix_vector_ctrl_t` { `kNETC_MsixIntrMaskBit` = 1U }

MSIX vector control field.

- enum `netc_tx_ext_flags_t` {
`kNETC_TxExtVlanInsert` = 0x1,
`kNETC_TxExtTwoStepTs` = 0x1 << 2U }

METC Extension Transmit Buffer Descriptor Extension flags field.

29.2 Data Structure Documentation

29.2.1 struct netc_vlan_t

Data Fields

- `uint32_t vid`: 12
Vlan Identifier.
- `uint32_t dei`: 1
Drop Eligible indicator.
- `uint32_t pcp`: 3
Priority.
- `uint32_t tpid`: 16
Tag protocol identifier.

Field Documentation

- (1) `uint32_t netc_vlan_t::vid`
- (2) `uint32_t netc_vlan_t::dei`
- (3) `uint32_t netc_vlan_t::pcp`
- (4) `uint32_t netc_vlan_t::tpid`

29.2.2 struct netc_buffer_struct_t

Driver can send/receive one frame spread across multiple buffers

Data Fields

- `void * buffer`
Buffer address.
- `uint16_t length`
Buffer data length.

Field Documentation

- (1) `void* netc_buffer_struct_t::buffer`
- (2) `uint16_t netc_buffer_struct_t::length`

29.2.3 struct netc_frame_struct_t**Data Fields**

- `netc_buffer_struct_t * buffArray`
Buffer array.
- `uint16_t length`
Buffer array length.

Field Documentation

- (1) `netc_buffer_struct_t* netc_frame_struct_t::buffArray`

Tx: [in]App sets, Rx: [in/out]App sets prepared array, driver sets back received buffers array.

- (2) `uint16_t netc_frame_struct_t::length`

Tx: [in]App sets, Rx: [in/out]App sets prepared array length, driver sets back received buffers array length.

29.2.4 struct netc_frame_attr_t**Data Fields**

- `bool isTsAvail`
Rx frame timestamp is available or not.
- `bool isVlanExtracted`
Rx frame VLAN header is available or not.
- `uint32_t timestamp`
The timestamp of this Rx frame.

Field Documentation

- (1) **bool netc_frame_attr_t::isTsAvail**
- (2) **bool netc_frame_attr_t::isVlanExtracted**
- (3) **uint32_t netc_frame_attr_t::timestamp**

29.2.5 struct netc_tx_frame_info_t**Data Fields**

- **bool isTsAvail**
Tx frame timestamp is available or not.
- **uint32_t timestamp**
The timestamp of this Tx frame, valid when isTsAvail is true.
- **bool isTxTsIdAvail**
Switch port Tx frame timestamp Identifier is available or not.
- **uint16_t txtsid**
The Transmit Timestamp Identifier, valid when isTsIdAvail is true, use for Switch management ENETC direct frame which has specified a timestamp request.
- **void *context**
Private context provided by the user.
- **netc_ep_tx_status_t status**
Transmit status.

Field Documentation

- (1) **bool netc_tx_frame_info_t::isTsAvail**
- (2) **uint32_t netc_tx_frame_info_t::timestamp**
- (3) **bool netc_tx_frame_info_t::isTxTsIdAvail**
- (4) **uint16_t netc_tx_frame_info_t::txtsid**
- (5) **void* netc_tx_frame_info_t::context**
- (6) **netc_ep_tx_status_t netc_tx_frame_info_t::status**

29.2.6 struct netc_msix_entry_t**Data Fields**

- **uint64_t msgAddr**
Message address.
- **uint32_t msgData**
Message data.
- **uint32_t control**

Vector control, netc_msix_vector_ctrl_t.

Field Documentation

- (1) `uint64_t netc_msix_entry_t::msgAddr`
- (2) `uint32_t netc_msix_entry_t::msgData`
- (3) `uint32_t netc_msix_entry_t::control`

29.3 Enumeration Type Documentation

29.3.1 anonymous enum

Enumerator

- kStatus_NETC_RxFrameEmpty* Rx BD ring empty.
- kStatus_NETC_RxTsrResp* Rx timestamp reference response.
- kStatus_NETC_RxFrameError* Rx frame error.
- kStatus_NETC_TxFrameOverLen* Tx frame over length.
- kStatus_NETC_LackOfResource* Lack of resources to configure certain features.
- kStatus_NETC_Unsupported* Unsupported operation/feature.
- kStatus_NETC_RxHRZeroFrame* Rx frame host reason is zero.
- kStatus_NETC_RxHRNotZeroFrame* Rx frame host reason is not zero.
- kStatus_NETC_NotFound* No entry found in hardware tables.
- kStatus_NETC_EntryExists* An entry already exists in hardware tables.

29.3.2 enum netc_ep_event_t

Enumerator

- kNETC_EPRxEvent* EP Rx interrupt event.
- kNETC_EPTxEvent* EP Tx interrupt event.

29.3.3 enum netc_ep_tx_status_t

Enumerator

- kNETC_EPTxSuccess* Success transmission.
- kNETC_EPTxProgramErr* Error exists in either the Tx BD, the Tx ring registers, or both.
- kNETC_EPTxTsdDrop* The time defined in TX_START expired before frame could be transmitted.
- kNETC_EPTxFrameSizeErr* Frame size error.
- kNETC_EPTxNullAddr* Null address.
- kNETC_EPTxInvalidLength* Invalid frame/buffer/chain length.

kNETC_EPTxSrcMacSpoofingDetect Source MAC address spoofing detected.
kNETC_EPTxPortRestDrop Frame dropped due to port reset.
kNETC_EPTxPortDisableDrop Frame dropped due to port disable.
kNETC_EPTxVlanTpIdDrop VLAN TPID not allowed.
kNETC_EPTxSmsParamErr Programming error in buffer descriptor used for direct switch enqueue.
kNETC_EPTxFrameGateErr Frame too large for time gating window.
kNETC_EPTxAxiReadErr AXI read error.
kNETC_EPTxAxiWriteErr AXI write error.
kNETC_EPTxMultiBitECCErr Frame not transmitted(dropped) due to a multi-bit ECC error detected.
kNETC_EPTxParityErr Parity error.
kNETC_EPTxSwCongestion Frame dropped due to switch congestion.

29.3.4 enum netc_vlan_tpid_select_t

Enumerator

kNETC_StanCvlan 0x8100.
kNETC_StanSvlan 0x88A8.
kNETC_CustomVlan1 CVLANR1[ETYPE].
kNETC_CustomVlan2 CVLANR2[ETYPE].

29.3.5 enum netc_msix_vector_ctrl_t

Enumerator

kNETC_MsixIntrMaskBit MSIX vector control interrupt mask bit.

29.3.6 enum netc_tx_ext_flags_t

Enumerator

kNETC_TxExtVlanInsert Enable VLAN insert.
kNETC_TxExtTwoStepTs Enable two-step timestamp offload.

29.4 API layer

The API layer is available for user access and is supposed to be the only interfaces for the user. A module in the API layer is not necessarily a direct mapping of a hardware peripheral but provides a higher functional model for the user.

For example, the Endpoint API module models the endpoint usage which covers the configuration of E-NETC peripheral, port peripheral, Station interface peripheral, etc... It abstracts the hardware details to provide a simpler model to interact on the network.

The modules APIs require a handle (`xxx_handle_t` pointer) which is used to get the appropriate context.

NETC is divided into 4 main groups:

- Endpoint: NETC works as endpoint
- Switch: NETC works as switch
- Timer: Timer functionalities
- MDIO: MDIO functionalities

29.5 Hardware layer

29.5.1 Overview

The hardware layer is mainly designed for code reuse. For example, in the API layer, Switch and Endpoint module which share common code such as port and station interface (SI). In general, the hardware layer maps closely to the hardware modules. User should not invoke the hardware layer directly.

Modules

- [Hardware Common Functions](#)
- [Hardware ENETC](#)
- [Hardware Port](#)
- [Hardware Port MAC](#)
- [Hardware Station Interface\(SI\)](#)
- [Hardware Switch](#)
- [Hardware Table Access Functions](#)

Data Structures

- struct [netc_isi_kc_rule_t](#)
NETC PSFP kc profile configuration, the key size (not include the spmp and portp) is up to 16 bytes.
[More...](#)
- struct [netc_vlan_classify_config_t](#)
NETC Vlan classification config. [More...](#)
- struct [netc_qos_classify_profile_t](#)
NETC Qos Classification profile file (vlan PCP/DEI to IPV/DR map) [More...](#)
- struct [netc_ipf_config_t](#)
NETC Ingress Filter config. [More...](#)
- struct [netc_func_t](#)
Register groups for the PCIe function. [More...](#)
- struct [netc_port_hw_t](#)
Register groups for the Port/Link hardware. [More...](#)
- struct [netc_enetc_hw_t](#)
Register group for the ENETC peripheral hardware. [More...](#)
- struct [netc_switch_hw_t](#)
Register group for the Switch peripheral hardware. [More...](#)
- struct [netc_timer_hw_t](#)
Register group for the Timer peripheral hardware. [More...](#)
- struct [netc_mdio_hw_t](#)
Register group for both EMDIO and port external MDIO. [More...](#)

Macros

- #define [getSiInstance\(si\) \(\(uint8_t\)\(\(uint16_t\)\(si\) >> 8U\)\)](#)
Get SI information from netc_hw_si_idx_t.
- #define [getSiNum\(si\) \(\(\(uint16_t\)\(si\) >> 4U\) & 0xFU\)\)](#)

- `#define getSiIdx(si) ((uint8_t)((uint16_t)(si)&0xFU))`
The actual index in the netc_hw_si_idx_t.
- `#define NETC_MSIX_TABLE_OFFSET (0x10000U)`
MSIX table address offset.
- `#define NETC_MSIX_TABLE_PBA_OFFSET (0x800U)`
MSIX PBA address offset.
- `#define NETC_NANOSECOND_ONE_SECOND (10000000000UL)`
Nanosecond in one second.

Enumerations

- enum `netc_hw_enetc_idx_t` {

`kNETC_ENETC0,`

`kNETC_ENETC1 }`

ENETC index enumerator.
- enum `netc_hw_switch_idx_t` { `kNETC_SWITCH0` }

SWITCH index enumerator.
- enum `netc_hw_port_idx_t` {

`kNETC_ENETC0Port = 0U,`

`kNETC_ENETC1Port = 0U,`

`kNETC_SWITCH0Port0 = 0U,`

`kNETC_SWITCH0Port1 = 1U,`

`kNETC_SWITCH0Port2 = 2U,`

`kNETC_SWITCH0Port3 = 3U,`

`kNETC_SWITCH0Port4 = 4U }`

Port Resource for the NETC module.
- enum `netc_hw_tc_idx_t` {

`kNETC_TxTC0 = 0,`

`kNETC_TxTC1,`

`kNETC_TxTC2,`

`kNETC_TxTC3,`

`kNETC_TxTC4,`

`kNETC_TxTC5,`

`kNETC_TxTC6,`

`kNETC_TxTC7 }`

Traffic class enumerator.
- enum `netc_hw_bdr_idx_t`

Enumeration for the ENETC SI BDR identifier.
- enum `netc_hw.swt_cbdr_idx_t` {

`kNETC_SWTCBDR0 = 0U,`

`kNETC_SWTCBDR1 }`

Switch command BD ring index enumerator.
- enum `netc_hw_etm_class_queue_idx_t` {

```
kNETC_ClassQueue0 = 0,
kNETC_ClassQueue1,
kNETC_ClassQueue2,
kNETC_ClassQueue3,
kNETC_ClassQueue4,
kNETC_ClassQueue5,
kNETC_ClassQueue6,
kNETC_ClassQueue7 }
```

Enumerator for ETM class queue identifier.

- enum `netc_hw_congestion_group_idx_t`

Enumerator for the ETM congestion group.

- enum `netc_hw_mii_mode_t` {

```
kNETC_XgmiiMode = 0U,
kNETC_MiiMode = 1U,
kNETC_GmiiMode = 2U,
kNETC_RmiiMode = 3U,
kNETC_RgmiiMode = 4U }
```

Defines the MII/RGMII mode for data interface between the MAC and the PHY.

- enum `netc_hw_mii_speed_t` {

```
kNETC_MiiSpeed10M = 0U,
kNETC_MiiSpeed100M = 1U,
kNETC_MiiSpeed1000M = 2U,
kNETC_MiiSpeed2500M = 3U,
kNETC_MiiSpeed5G = 4U,
kNETC_MiiSpeed10G = 5U }
```

*Defines the speed for the *MII data interface.*

- enum `netc_hw_mii_duplex_t` {

```
kNETC_MiiHalfDuplex = 0U,
kNETC_MiiFullDuplex }
```

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

29.5.2 Data Structure Documentation

29.5.2.1 struct `netc_isi_kc_rule_t`

Data Fields

- bool `etp`: 1
2 Byte Ethertype field present in the key
- bool `sqtpt`: 1
1 Byte Sequence Tag present in the key
- bool `ipcpp`: 1
inner VLAN header's PCP field present in the key
- bool `ividp`: 1
inner VLAN ID present in the key
- bool `opcpp`: 1

- bool **ovidp**: 1
outer VLAN header's PCP field present in the key
- bool **smacp**: 1
outer VLAN ID present in the key
- bool **dmacp**: 1
6 bytes of source MAC address present in the key
- bool **spmp**: 1
6 bytes of destination MAC address present in the key
- bool **portp**: 1
switch port masquerading flag present in the key
- bool **valid**: 1
Key Construction is valid.
- uint8_t **lbMask**: 3
Payload Last Byte Mask.
- uint8_t **fbMask**: 3
Payload First Byte Mask.
- uint8_t **byteOffset**: 7
Payload Byte Offset where field extraction begins.
- uint8_t **numBytes**: 4
Specify the size (numBytes + 1) of the payload key field.
- uint8_t **pfp**: 1
Payload field Present.

29.5.2.2 struct netc_vlan_classify_config_t

Data Fields

- bool **enableCustom1**
Enable/Disable custom0 ether type.
- uint16_t **custom1EtherType**
Ethertype.
- bool **enableCustom2**
Enable/Disable custom0 ether type.
- uint16_t **custom2EtherType**
Ethertype.
- uint16_t **preStandRTAGType**
802.1CB draft 2.0 R-TAG EtherType value.

Field Documentation

(1) uint16_t netc_vlan_classify_config_t::preStandRTAGType

PSRTAGETR. Only applicable for switch

29.5.2.3 struct netc_qos_classify_profile_t

Data Fields

- uint8_t `ipv` [16]
Index is created from PCP (3 bits) + DEI (1 bit) field.
- uint8_t `dr` [16]
Index is created from PCP (3 bits) + DEI (1 bit) field.

Field Documentation

(1) `uint8_t netc_qos_classify_profile_t::ipv[16]`

Value is the mapped IPV for Qos.

(2) `uint8_t netc_qos_classify_profile_t::dr[16]`

Value is the mapped DR for QoS.

29.5.2.4 struct netc_ipf_config_t

Data Fields

- bool `I2DiscardMCsmac`
DOSL2CR.
- bool `I2DiscardSmacEquDmac`
DOSL2CR.
- bool `I3DiscardSipEquDip`
DOSL3CR.

Field Documentation

(1) `bool netc_ipf_config_t::I2DiscardMCsmac`

Discard received frames with Multicast SMAC address

(2) `bool netc_ipf_config_t::I2DiscardSmacEquDmac`

Discard received frames with SMAC = DMAC

(3) `bool netc_ipf_config_t::I3DiscardSipEquDip`

Discard IPV3/IPV6 source address == destination address

29.5.2.5 struct netc_func_t

29.5.2.6 struct netc_port_hw_t

Data Fields

- NETC_PORT_Type * **port**
Port Address.
- NETC_ETH_LINK_Type * **eth**
MAC Port Address.
- NETC_PSEUDO_LINK_Type * **pseudo**
Pseudo link port address.

29.5.2.7 struct netc_enetc_hw_t

Data Fields

- netc_func_t **func**
PCIE function register.
- NETC_ENETC_Type * **base**
Base register of ENETC module.
- NETC_SW_ENETC_Type * **common**
Common register of ENETC module.
- netc_port_hw_t **portGroup**
Port register group.
- ENETC_GLOBAL_Type * **global**
Global NETC address.
- ENETC_SI_Type * **si**
Station Interface for the P/V SI.
- netc_msix_entry_t * **msixTable**
MSIX table address.

29.5.2.8 struct netc_switch_hw_t

Data Fields

- ENETC_PCI_TYPE0_Type * **func**
Base address for PCIE function.
- NETC_SW_Type * **base**
Base address for Switch.
- NETC_SW_ENETC_Type * **common**
Common address of ENETC Module.
- netc_port_hw_t **ports** [FSL_FEATURE_NETC_SWITCH_MAX_PORT_NUMBER]
Port Register Group.
- ENETC_GLOBAL_Type * **global**
Global NETC Register Base Address.
- netc_msix_entry_t * **msixTable**
MSIX table address.

29.5.2.9 struct netc_timer_hw_t

Data Fields

- ENETC_PCI_TYPE0_Type * **func**
PCIE function register.
- ENETC_PF_TMR_Type * **base**
Base register address for timer module.
- ENETC_GLOBAL_Type * **global**
Global NETC register address.
- netc_msix_entry_t * **msixTable**
MSIX table address.

29.5.2.10 struct netc_mdio_hw_t

Data Fields

- __IO uint32_t **EMDIO_CFG**
External MDIO configuration register, offset: 0x1C00.
- __IO uint32_t **EMDIO_CTL**
External MDIO interface control register, offset: 0x1C04.
- __IO uint32_t **EMDIO_DATA**
External MDIO interface data register, offset: 0x1C08.
- __IO uint32_t **EMDIO_ADDR**
External MDIO register address register, offset: 0x1C0C.
- __I uint32_t **EMDIO_STAT**
External MDIO status register, offset: 0x1C10.
- __IO uint32_t **PHY_STATUS_CFG**
PHY status configuration register, offset: 0x1C20.
- __IO uint32_t **PHY_STATUS_CTL**
PHY status control register, offset: 0x1C24.
- __I uint32_t **PHY_STATUS_DATA**
PHY status data register, offset: 0x1C28.
- __IO uint32_t **PHY_STATUS_ADDR**
PHY status register address register, offset: 0x1C2C.
- __IO uint32_t **PHY_STATUS_EVENT**
PHY status event register, offset: 0x1C30.
- __IO uint32_t **PHY_STATUS_MASK**
PHY status mask register, offset: 0x1C34.

29.5.3 Macro Definition Documentation

29.5.3.1 #define getSiInstance(si) ((uint8_t)((uint16_t)(si) >> 8U))

The ENETC instance of this SI.

29.5.3.2 `#define getSiNum(si) (((uint16_t)(si) >> 4U) & 0xFU))`

29.5.3.3 `#define getSidx(si) (((uint16_t)(si)&0xFU))`

29.5.3.4 `#define NETC_MSIX_TABLE_OFFSET (0x10000U)`

29.5.3.5 `#define NETC_MSIX_TABLE_PBA_OFFSET (0x800U)`

29.5.3.6 `#define NETC_NANOSECOND_ONE_SECOND (1000000000UL)`

29.5.4 Enumeration Type Documentation

29.5.4.1 enum netc_hw_enetc_idx_t

Enumerator

kNETC_ENETC0 ENETC hardware 0.

kNETC_ENETC1 ENETC hardware 0.

29.5.4.2 enum netc_hw_switch_idx_t

Enumerator

kNETC_SWITCH0 SWITCH hardware 0.

29.5.4.3 enum netc_hw_port_idx_t

Enumerator

kNETC_ENETC0Port MAC port for ENETC0.

kNETC_ENETC1Port Pseudo MAC port for ENETC1.

kNETC_SWITCH0Port0 MAC port0 for SWITCH.

kNETC_SWITCH0Port1 MAC port1 for SWITCH.

kNETC_SWITCH0Port2 MAC port2 for SWITCH.

kNETC_SWITCH0Port3 MAC port3 for SWITCH.

kNETC_SWITCH0Port4 Pseudo port4 for SWITCH.

29.5.4.4 enum netc_hw_tc_idx_t

Enumerator

kNETC_TxTC0 Traffic class 0.

kNETC_TxTC1 Traffic class 1.

kNETC_TxTC2 Traffic class 2.
kNETC_TxTC3 Traffic class 3.
kNETC_TxTC4 Traffic class 4.
kNETC_TxTC5 Traffic class 5.
kNETC_TxTC6 Traffic class 6.
kNETC_TxTC7 Traffic class 7.

29.5.4.5 enum netc_hw_swt_cbdr_idx_t

Enumerator

kNETC_SWTCBDR0 Switch command BD ring 0.
kNETC_SWTCBDR1 Switch command BD ring 1.

29.5.4.6 enum netc_hw_etm_class_queue_idx_t

Enumerator

kNETC_ClassQueue0 ETM Class Queue 0.
kNETC_ClassQueue1 ETM Class Queue 1.
kNETC_ClassQueue2 ETM Class Queue 2.
kNETC_ClassQueue3 ETM Class Queue 3.
kNETC_ClassQueue4 ETM Class Queue 4.
kNETC_ClassQueue5 ETM Class Queue 5.
kNETC_ClassQueue6 ETM Class Queue 6.
kNETC_ClassQueue7 ETM Class Queue 7.

29.5.4.7 enum netc_hw_mii_mode_t

Enumerator

kNETC_XgmiiMode XGMII mode for data interface.
kNETC_MiiMode MII mode for data interface.
kNETC_GmiiMode GMII mode for data interface.
kNETC_RmiiMode RMII mode for data interface.
kNETC_RgmiiMode RGMII mode for data interface.

29.5.4.8 enum netc_hw_mii_speed_t

Enumerator

kNETC_MiiSpeed10M Speed 10 Mbps.

kNETC_MiiSpeed100M Speed 100 Mbps.
kNETC_MiiSpeed1000M Speed 1000 Mbps.
kNETC_MiiSpeed2500M Speed 2500 Mbps.
kNETC_MiiSpeed5G Speed 5Gbps.
kNETC_MiiSpeed10G Speed 10Gbps Mbps.

29.5.4.9 enum netc_hw_mii_duplex_t

Enumerator

kNETC_MiiHalfDuplex Half duplex mode.
kNETC_MiiFullDuplex Full duplex mode.

29.5.5 Hardware Station Interface(SI)

29.5.5.1 Overview

Group for Station Interface(SI) API and data structure. data structures are prefixed with netc_si. APIs are prefixed with NETC_SI.

Data Structures

- struct `netc_hw_enetc_si_config_t`
Station Interface configuration. [More...](#)
- struct `netc_si_l2mf_config_t`
L2 Mac Filter Configuration for SI. [More...](#)
- struct `netc_si_l2vf_config_t`
L2 VLAN Filter Configuration for SI. [More...](#)
- struct `netc_si_discard_statistic_t`
SI frame drop statistic struct. [More...](#)
- struct `netc_si_traffic_statistic_t`
SI traffic statistic struct. [More...](#)
- struct `netc_si_config_t`
SI Configuration. [More...](#)
- union `netc_tx_bd_t`
Transmit Buffer Descriptor format. [More...](#)
- union `netc_rx_bd_t`
Receive Buffer Descriptor format. [More...](#)
- struct `netc_tx_bdr_config_t`
Configuration for the SI Tx Buffer Descriptor Ring Configuration. [More...](#)
- struct `netc_tx_bdr_t`
Transmit BD ring handler data structure. [More...](#)
- struct `netc_rx_bdr_config_t`
Configuration for the SI Rx Buffer Descriptor Ring Configuration. [More...](#)
- struct `netc_rx_bdr_t`
Receive BD ring handler data structure. [More...](#)
- struct `netc_bdr_config_t`
Configuration for the buffer descriptors ring. [More...](#)
- struct `netc_psi_rx_msg_t`
PSI receive message information. [More...](#)
- struct `netc_vsi_msg_tx_status_t`
VSI message transmit status. [More...](#)

Macros

- #define `NETC_VLAN_PCP_DEI_VALUE(pcp, dei)` (((uint8_t)((pcp)&0x7U) << 1U) | ((dei)&0x1U))
Macro to cover VLAN PCP, DEI value to internal used pcpDei value.

Enumerations

- enum `netc_hw_enetc_si_vlan_type` {

 kNETC_ENETC_StaCVlan = 0x1U,

 kNETC_ENETC_StaSVlan = 0x2U,

 kNETC_ENETC_CustomVlan1 = 0x4U,

 kNETC_ENETC_CustomVlan2 = 0x8U }

 VLAN Ethertypes.
- enum `netc_hw_enetc_si_rxr_group` {

 kNETC_SiBDRGroupOne = 0U,

 kNETC_SiBDRGroupTwo }
- SI receive BD ring group index.*
- enum `netc_tx_bdr_flags_t`

 Status/Interrupts flags for the TX BDR.
- enum `netc_rx_bdr_flags_t`

 Status/Interrupts flags for the RX BDR.
- enum `netc_psi_msg_flags_t` {

 kNETC_PsiRxMsgFromVsi1Flag = 0x2,

 kNETC_PsiFLRFromVsi1Flag = 0x20000 }

 PSI message interrupt type.
- enum `netc_vsi_msg_flags_t` {

 kNETC_VsiMsgTxFlag = 0x1,

 kNETC_VsiMsgRxFlag = 0x1 << 1U }

 VSI message interrupt flags.
- enum `netc_vsi_number_t`

 VSI number bit map, VSII starts from bit1.

29.5.5.2 Data Structure Documentation

29.5.5.2.1 struct `netc_hw_enetc_si_config_t`

Data Fields

- uint32_t `bandWeight`: 4

 Station interface traffic class bandwidth weight.
- uint32_t `vlanCtrl`: 4

 VLAN Ethertypes can be inserted by the SI driver, set with OR of `netc_hw_enetc_si_vlan_type`.
- uint32_t `antiSpoofEnable`: 1

 Anti-spoofing enable.
- uint32_t `vlanInsertEnable`: 1

 Software SI-based VLAN Insertion enable, avtive when `enSIBaseVlan` is true.
- uint32_t `vlanExtractEnable`: 1

 SI-based VLAN removed from frame enable, avtive when `enSIBaseVlan` is true.
- uint32_t `sourcePruneEnable`: 1

 Source pruning enable.
- uint32_t `rxRingUse`: 7

 Number of Rx Rings to be used, when enable Rx ring group, this equal to the sum of all Rx group rings.
- uint32_t `txRingUse`: 7

Number of Tx Rings to be used, note that when SI is Switch management ENETC SI, the number not include Tx ring 0.

- **uint32_t valnToIpvEnable:** 1
 - Enable the VLAN PCP/DEI value (use NETC_VLAN_PCP_DEI_VALUE macro) to internal priority value mapping.*
- **uint32_t rxBdrGroupNum:** 2
 - Rx BD ring group number, range in 0 ~ 2.*
- **uint32_t ringPerBdrGroup:** 3
 - The ring number in every Rx BD ring group, range in 1 ~ 8, active when rxBdrGroupNum not equal zero.*
- **netc_hw_enetc_si_rxr_group defaultRxBdrGroup**
 - The selected Rx BD ring group, active when rxBdrGroupNum not equal zero.*
- **uint8_t vlanToIpvMap [16]**
 - Frame VLAN pcp|dei to IPV mapping, active when valnToIpvEnable is true.*
- **uint8_t ipvToRingMap [8]**
 - BD ring used within the default Rx BD ring group for IPV n, active when rxBdrGroupNum not equal zero.*
- **uint8_t vsiTcToTC [8]**
 - Maps the VSI traffic class to transmit traffic class, done after the ENETC txPrio to TC mapping, only available for VSI.*
- **bool enSIBaseVlan**
 - Enable use SI-based VLAN information.*
- **netc_enetc_vlan_tag_t siBaseVlan**
 - SI-based VLAN information, active when enSIBaseVlan is true.*

Field Documentation

- (1) `uint32_t netc_hw_enetc_si_config_t::vlanCtrl`
- (2) `uint32_t netc_hw_enetc_si_config_t::rxRingUse`
- (3) `uint32_t netc_hw_enetc_si_config_t::txRingUse`
- (4) `uint32_t netc_hw_enetc_si_config_t::valnTolpvEnable`
- (5) `uint32_t netc_hw_enetc_si_config_t::rxBdrGroupNum`
- (6) `uint32_t netc_hw_enetc_si_config_t::ringPerBdrGroup`
- (7) `netc_hw_enetc_si_rxr_group netc_hw_enetc_si_config_t::defaultRxBdrGroup`
- (8) `uint8_t netc_hw_enetc_si_config_t::vlanTolpvMap[16]`
- (9) `uint8_t netc_hw_enetc_si_config_t::ipvToRingMap[8]`
- (10) `uint8_t netc_hw_enetc_si_config_t::vsiTcToTC[8]`
- (11) `bool netc_hw_enetc_si_config_t::enSIBaseVlan`
- (12) `netc_enetc_vlan_tag_t netc_hw_enetc_si_config_t::siBaseVlan`

29.5.5.2.2 struct netc_si_l2mf_config_t

Data Fields

- `bool macUCPromis`
Enable/Disable MAC unicast promiscuous.
- `bool macMCPromis`
Enable/Disable MAC multicast promiscuous.
- `bool rejectUC`
Reject Unicast.
- `bool rejectMC`
Reject Multicast.
- `bool rejectBC`
Reject Broadcast.

Field Documentation

- (1) **bool netc_si_l2mf_config_t::macUCPromis**
- (2) **bool netc_si_l2mf_config_t::macMCPromis**
- (3) **bool netc_si_l2mf_config_t::rejectUC**
- (4) **bool netc_si_l2mf_config_t::rejectMC**
- (5) **bool netc_si_l2mf_config_t::rejectBC**

29.5.5.2.3 struct netc_si_l2vf_config_t**Data Fields**

- **bool acceptUntagged**
Accept/Reject untagged frame.
- **bool enPromis**
Enable/Disable VLAN promiscuous.
- **bool useOuterVlanTag**
Use outer/inner VLAN tag for filtering.

Field Documentation

- (1) **bool netc_si_l2vf_config_t::acceptUntagged**
- (2) **bool netc_si_l2vf_config_t::enPromis**
- (3) **bool netc_si_l2vf_config_t::useOuterVlanTag**

29.5.5.2.4 struct netc_si_discard_statistic_t**Data Fields**

- **uint32_t programError**
Due to programming error (non-existing BD ring or non-existing group, or SI disabled or BD ring disabled).
- **uint32_t busError**
Due to system bus error.
- **uint32_t lackBD [14]**
Due to lack of Rx BDs available.

Field Documentation

- (1) `uint32_t netc_si_discard_statistic_t::programError`
- (2) `uint32_t netc_si_discard_statistic_t::busError`
- (3) `uint32_t netc_si_discard_statistic_t::lackBD[14]`

29.5.5.2.5 struct netc_si_traffic_statistic_t**29.5.5.2.6 struct netc_si_config_t****Data Fields**

- `uint32_t tcBWWeight`
SI traffic class bandwidth weight.

Field Documentation

- (1) `uint32_t netc_si_config_t::tcBWWeight`

29.5.5.2.7 union netc_tx_bd_t

A union type cover the BD used as Standard/Extended/WriteBack format.

Field Documentation

- (1) `uint64_t netc_tx_bd_t::addr`

Little Endian.

- (2) `uint16_t netc_tx_bd_t::bufLen`
- (3) `uint16_t netc_tx_bd_t::frameLen`
- (4) `uint32_t netc_tx_bd_t::flags`
- (5) `uint32_t netc_tx_bd_t::enableInterrupt`
- (6) `uint32_t netc_tx_bd_t::isExtended`
- (7) `uint32_t netc_tx_bd_t::isFinal`
- (8) `uint32_t netc_tx_bd_t::timestamp`

Timestamp write back.

- (9) `uint32_t netc_tx_bd_t::__pad0__`

/ Transmit timestamp identifier, only active on Switch management ENETC.

- (10) `uint16_t netc_tx_bd_t::tpid`
- (11) `uint16_t netc_tx_bd_t::vid`
- (12) `uint16_t netc_tx_bd_t::dei`
- (13) `uint16_t netc_tx_bd_t::pcp`
- (14) `uint8_t netc_tx_bd_t::eFlags`
- (15) `uint8_t netc_tx_bd_t::isFinal`
- (16) `uint32_t netc_tx_bd_t::status`
- (17) `uint32_t netc_tx_bd_t::written`

29.5.5.2.8 union netc_rx_bd_t

Field Documentation

- (1) `uint64_t netc_rx_bd_t::addr`
- (2) `uint16_t netc_rx_bd_t::internetChecksum`
- (3) `uint16_t netc_rx_bd_t::parserSummary`
- (4) `uint32_t netc_rx_bd_t::srcPort`
- (5) `uint32_t netc_rx_bd_t::rssHash`
- (6) `uint32_t netc_rx_bd_t::rssHashSwt`
- (7) `uint16_t netc_rx_bd_t::bufLen`
- (8) `uint16_t netc_rx_bd_t::vid`
- (9) `uint16_t netc_rx_bd_t::dei`
- (10) `uint16_t netc_rx_bd_t::pcp`
- (11) `uint8_t netc_rx_bd_t::tpid`
- (12) `uint8_t netc_rx_bd_t::hr`
- (13) `uint8_t netc_rx_bd_t::flags`
- (14) `uint8_t netc_rx_bd_t::error`
- (15) `uint8_t netc_rx_bd_t::isReady`
- (16) `uint8_t netc_rx_bd_t::isFinal`
- (17) `uint32_t netc_rx_bd_t::timestamp`

- (18) `uint16_t netc_rx_bd_t::txtsid`
- (19) `uint32_t netc_rx_bd_t::hr`
- (20) `uint32_t netc_rx_bd_t::isReady`
- (21) `uint32_t netc_rx_bd_t::isFinal`

29.5.5.2.9 struct netc_tx_bdr_config_t

Data Fields

- `uint32_t len`
Size of BD ring which shall be multiple of 8 BD.
- `netc_tx_bd_t * bdArray`
BDR base address which shall be 128 bytes aligned.
- `netc_tx_frame_info_t * dirtyArray`
Tx cleanup ring.
- `bool enIntr`
Enable/Disable completion interrupt.
- `bool enThresIntr`
Enable/Disable threshold interrupt.
- `bool enCoalIntr`
Enable/Disable interrupt coalescing.
- `uint32_t intrThreshold`
Interrupt coalescing packet threshold.
- `uint32_t intrTimerThres`
Interrupt coalescing timer threshold, specified in NETC clock cycles.
- `uint8_t msixEntryIdx`
MSIX entry index of Tx ring interrupt.
- `bool isVlanInsert`
Enable/Disable VLAN insert offload.
- `bool isUserCRC`
Enable/Disable user provided the CRC32 - FCS at end of frame.
- `uint8_t wrrWeight`
Weight used for arbitration when rings have same priority.
- `uint8_t priority`
Priority of the Tx BDR.

Field Documentation

- (1) `uint32_t netc_tx_bdr_config_t::len`
- (2) `netc_tx_bd_t* netc_tx_bdr_config_t::bdArray`
- (3) `netc_tx_frame_info_t* netc_tx_bdr_config_t::dirtyArray`
- (4) `bool netc_tx_bdr_config_t::enIntr`
- (5) `bool netc_tx_bdr_config_t::enThresIntr`
- (6) `bool netc_tx_bdr_config_t::enCoallIntr`
- (7) `uint32_t netc_tx_bdr_config_t::intrThreshold`
- (8) `uint32_t netc_tx_bdr_config_t::intrTimerThres`
- (9) `uint8_t netc_tx_bdr_config_t::msixEntryIdx`
- (10) `bool netc_tx_bdr_config_t::isVlanInsert`
- (11) `bool netc_tx_bdr_config_t::isUserCRC`
- (12) `uint8_t netc_tx_bdr_config_t::wrrWeight`
- (13) `uint8_t netc_tx_bdr_config_t::priority`

29.5.5.2.10 struct netc_tx_bdr_t**Data Fields**

- `netc_tx_bd_t * bdBase`
Tx BDR base address.
- `netc_tx_frame_info_t * dirtyBase`
Tx cleanup ring base address.
- `uint16_t producerIndex`
Current index for transmit.
- `uint16_t cleanIndex`
Current index for tx cleaning.
- `uint32_t len`
Length of this BD ring.

Field Documentation

- (1) `netc_tx_bd_t* netc_tx_bdr_t::bdBase`
- (2) `netc_tx_frame_info_t* netc_tx_bdr_t::dirtyBase`
- (3) `uint16_t netc_tx_bdr_t::producerIndex`
- (4) `uint16_t netc_tx_bdr_t::cleanIndex`
- (5) `uint32_t netc_tx_bdr_t::len`

29.5.5.2.11 struct netc_rx_bdr_config_t

Data Fields

- bool `extendDescEn`
False - Use 16Bytes standard BD.
- `netc_rx_bd_t * bdArray`
BD ring base address which shall be 128 bytes aligned.
- `uint32_t len`
BD ring length in the unit of 16Bytes standard BD.
- `uint64_t * buffAddrArray`
Rx buffers array with BD length(half of BD length if use exteneded BD).
- `uint16_t bufferSize`
Size of all Rx buffers in this BD ring.
- bool `enThresIntr`
Enable/Disable threshold interrupt.
- bool `enCoalIntr`
Enable/Disable interrupt coalescing.
- `uint32_t intrThreshold`
Interrupt coalescing packet threshold.
- `uint32_t intrTimerThres`
Interrupt coalescing timer threshold, specified in NETC clock cycles.
- `uint8_t msixEntryIdx`
MSIX entry index of Rx ring interrupt.
- bool `disVlanPresent`
Disable/Enable VLAN in BD.
- bool `enVlanExtract`
Enable/Disable VLAN extract.
- bool `isKeepCRC`
Whether user provided the CRC32 - FCS at end of frame.
- bool `congestionMode`
False - lossy.
- bool `enHeaderAlign`
Enable/disable +2B alignment to frame.

Field Documentation(1) **bool netc_rx_bdr_config_t::extendDescEn**

True - Use 32Bytes extended BD.

(2) **netc_rx_bd_t* netc_rx_bdr_config_t::bdArray**(3) **uint32_t netc_rx_bdr_config_t::len**

Shall be multiple of 8/16 for standard/exteneded BD.

(4) **uint64_t* netc_rx_bdr_config_t::buffAddrArray**(5) **uint16_t netc_rx_bdr_config_t::buffSize**(6) **bool netc_rx_bdr_config_t::enThresIntr**(7) **bool netc_rx_bdr_config_t::enCoallIntr**(8) **uint32_t netc_rx_bdr_config_t::intrThreshold**(9) **uint32_t netc_rx_bdr_config_t::intrTimerThres**(10) **uint8_t netc_rx_bdr_config_t::msixEntryIdx**(11) **bool netc_rx_bdr_config_t::disVlanPresent**(12) **bool netc_rx_bdr_config_t::enVlanExtract**(13) **bool netc_rx_bdr_config_t::isKeepCRC**(14) **bool netc_rx_bdr_config_t::congestionMode**

True - lossless.

(15) **bool netc_rx_bdr_config_t::enHeaderAlign****29.5.5.2.12 struct netc_rx_bdr_t****Data Fields**

- **netc_rx_bd_t * bdBase**
Rx BDR base address.
- **bool extendDesc**
Use extended buffer descriptor.
- **uint16_t index**
Current index for read.
- **uint32_t len**
Length of this BD ring, unit of 16Bytes standard BD.
- **uint64_t * buffArray**

- **Rx buffers array of this ring.**
- **uint32_t bufferSize**
Rx buffers size for all BDs in this ring.

Field Documentation

- (1) **netc_rx_bd_t* netc_rx_bdr_t::bdBase**
- (2) **bool netc_rx_bdr_t::extendDesc**
- (3) **uint16_t netc_rx_bdr_t::index**
- (4) **uint32_t netc_rx_bdr_t::len**
- (5) **uint64_t* netc_rx_bdr_t::buffArray**
- (6) **uint32_t netc_rx_bdr_t::buffSize**

29.5.5.2.13 struct netc_bdr_config_t

Data Fields

- **netc_rx_bdr_config_t * rxBdrConfig**
Receive buffer ring configuration array.
- **netc_tx_bdr_config_t * txBdrConfig**
Transmit buffer ring configuration array.

Field Documentation

- (1) **netc_rx_bdr_config_t* netc_bdr_config_t::rxBdrConfig**
- (2) **netc_tx_bdr_config_t* netc_bdr_config_t::txBdrConfig**

29.5.5.2.14 struct netc_psi_rx_msg_t

Data Fields

- **uint8_t * msgBuff**
The buffer address application set before receiving message.
- **uint32_t msgLen**
Received message length.

Field Documentation(1) `uint8_t* netc_psi_rx_msg_t::msgBuff`(2) `uint32_t netc_psi_rx_msg_t::msgLen`**29.5.5.2.15 struct netc_vsi_msg_tx_status_t****Data Fields**

- `bool txBusy`
The VSI Tx busy flag, become idle when the PSI receive and clear the related status.
- `bool isTxErr`
Tx error flag.
- `uint16_t msgCode`
The error code or user-defined content.

Field Documentation(1) `bool netc_vsi_msg_tx_status_t::txBusy`(2) `bool netc_vsi_msg_tx_status_t::isTxErr`(3) `uint16_t netc_vsi_msg_tx_status_t::msgCode`**29.5.5.3 Macro Definition Documentation**

29.5.5.3.1 #define NETC_VLAN_PCP_DEI_VALUE(*pcp, dei*) (((uint8_t)((*pcp*)&0x7U) << 1U) | ((*dei*)&0x1U))

29.5.5.4 Enumeration Type Documentation**29.5.5.4.1 enum netc_hw_enetc_si_vlan_type**

Enumerator

kNETC_ENETC_StanCVlan Standard C-VLAN 0x8100.*kNETC_ENETC_StanSVlan* Standard S-VLAN 0x88A8.*kNETC_ENETC_CustomVlan1* Custom VLAN as defined by CVLANR1[ETYPE].*kNETC_ENETC_CustomVlan2* Custom VLAN as defined by CVLANR2[ETYPE].**29.5.5.4.2 enum netc_hw_enetc_si_rxr_group**

Enumerator

kNETC_SiBDRGroupOne SI Rx BD ring group index one.*kNETC_SiBDRGroupTwo* SI Rx BD ring group index two.

29.5.5.4.3 enum netc_tx_bdr_flags_t

Each flag get its own bit thus it support bit AND/OR operation.

29.5.5.4.4 enum netc_rx_bdr_flags_t

Each flag get its own bit thus it support bit AND/OR operation.

29.5.5.4.5 enum netc_psi_msg_flags_t

Enumerator

kNETC_PsiRxMsgFromVsi1Flag Message receive interrupt enable, initiated by VSI1.

kNETC_PsiFLRFromVsi1Flag Function level reset interrupt enable, initiated by VSI1.

29.5.5.4.6 enum netc_vsi_msg_flags_t

Enumerator

kNETC_VsiMsgTxFlag Message sent to PSI has completed and response received.

kNETC_VsiMsgRxFlag Message received from PSI.

29.5.6 Hardware ENETC

29.5.6.1 Overview

Group for ENETC API and data structure. data structure with netc_enetc as part in the naming. API starts with NETC_Enetc.

Data Structures

- struct `netc_enetc_vlan_tag_t`
ENETC Port outer/inner VLAN tag. [More...](#)
- struct `netc_enetc_port_discard_statistic_t`
PORT discard count statistic. [More...](#)
- struct `netc_enetc_native_vlan_config_t`
ENETC Port outer/inner native VLAN config. [More...](#)
- struct `netc_enetc_parser_config_t`
ENETC parser configuration. [More...](#)

29.5.6.2 Data Structure Documentation

29.5.6.2.1 struct netc_enetc_vlan_tag_t

Data Fields

- `uint16_t pcp`: 3
Priority code point.
- `uint16_t dei`: 1
Drop eligible indicator.
- `uint16_t vid`: 12
VLAN identifier.
- `netc_vlan_tpid_select_t tpid`: 2
Tag protocol identifier.

29.5.6.2.2 struct netc_enetc_port_discard_statistic_t

Data Fields

- `uint32_t ingressDR [4]`
Discard count for port ingress congestion different DR.
- `uint32_t broadcastReject`
Broadcast frame drops count due to all SI enable broadcast reject.
- `uint32_t smacPruning`
Frames discard count due to port MAC source address pruning.
- `uint32_t unicastMacFilt`
Unicast frame discard count due to port MAC filtering.
- `uint32_t multicastMacFilt`
Multicast frame discard count due to MAC filtering.

- `uint32_t unicastVlanFilt`
Unicast frame discard count due to VLAN filtering.
- `uint32_t multicastVlanFilt`
Multicast frame discard count due to VLAN filtering.
- `uint32_t broadcastVlanFilt`
Broadcast frame discard count due to VLAN filtering.

29.5.6.2.3 struct netc_enetc_native_vlan_config_t

Data Fields

- `bool enUnderZeroVid`
Enable use the port default VLAN VID when the VID in the packet's is zero.
- `bool enUnderNoVlan`
Enable use the port default VLAN VID when the VLAN tag is not present.
- `netc_enetc_vlan_tag_t vlanTag`
Port native outer/inner VLAN tag, valid when enUnderZeroVid or enUnderNoVlan is true.

29.5.6.2.4 struct netc_enetc_parser_config_t

Data Fields

- `bool disL3Checksum`
Disable Layer 3 IPv4 Header checksum validation.
- `bool disL4Checksum`
Disable Layer 4 TCP and UDP checksum validation.
- `uint16_t etype`
Custom Ethertype value.
- `bool en`
Enables the detection and mapping.
- `uint8_t cp`
This value is mapped to the parse summary as a Non IP code point.

Field Documentation

- (1) `bool netc_enetc_parser_config_t::disL3Checksum`
- (2) `bool netc_enetc_parser_config_t::disL4Checksum`
- (3) `uint16_t netc_enetc_parser_config_t::etype`

Upon detecting this ether type the associated code point will be mapped to the parse summary as a Non IP code point.

- (4) `bool netc_enetc_parser_config_t::en`
- (5) `uint8_t netc_enetc_parser_config_t::cp`

29.5.7 Hardware Port

29.5.7.1 Overview

Group for PORT features API and data structure. data structures are prefixed with netc_port. APIs are prefixed with NETC_Port.

Modules

- [Hardware Port Rx](#)
- [Hardware Port Tx](#)

Data Structures

- [`struct netc_port_qos_mode_t`](#)
Port Qos mode. [More...](#)
- [`struct netc_port_parser_config_t`](#)
Port Parser config. [More...](#)
- [`struct netc_port_tg_config_t`](#)
Port time gate config. [More...](#)
- [`struct netc_port_tc_cbs_config_t`](#)
Configuration for the Credit Based Shaped for port TC. [More...](#)
- [`struct netc_port_tx_tc_config_t`](#)
Configuration for the port Tx Traffic Class. [More...](#)
- [`struct netc_port_discard_statistic_t`](#)
Switch or ENETC port Tx/Rx/Bridge discard statistic / reason. [More...](#)
- [`struct netc_port_vlan_classify_config_t`](#)
Port accepted Vlan classification config. [More...](#)
- [`struct netc_port_qos_classify_config_t`](#)
Port Qos Classification Config. [More...](#)
- [`struct netc_port_ipf_config_t`](#)
Port Ingress Filter Config. [More...](#)
- [`struct netc_port_psfp_isi_config`](#)
PSFP port config. [More...](#)
- [`struct netc_port_common_t`](#)
Port common configuration. [More...](#)

Enumerations

- [`enum netc_port_tgsl_status_t {`](#)
`kNETC_OperListActive = 0x1U,`
`kNETC_AdminListPending }`
Port time gate scheduling gate list status.
- [`enum netc_port_discard_tpye_t {`](#)
`kNETC_RxDiscard = 0U,`
`kNETC_TxDiscard,`
`kNETC_BridgeDiscard }`

- Port Tx/Rx discard counter in the datapath processing pipeline or bridge forwarding processing function.
- enum `netc_port_tpidlist_t` {

kNETC_OuterStanCvlan = 0x1U,

kNETC_OuterStanSvlan = 0x2U,

kNETC_OuterCustomVlan1 = 0x4U,

kNETC_OuterCustomVlan2 = 0x8U,

kNETC_InnerStanCvlan = 0x1U,

kNETC_InnerStanSvlan = 0x2U,

kNETC_InnerCustomVlan1 = 0x4U,

kNETC_InnerCustomVlan2 = 0x8U }

Defines Port TPID acceptance.
- enum `netc_port_ts_select_t` {

kNETC_SyncTime = 0U,

kNETC_FreeRunningTime }

Defines port timestamp selection.
- enum `netc_hw_preemption_mode_t` {

kNETC_PreemptDisable = 0U,

kNETC_PreemptOn64B,

kNETC_PreemptOn4B }

Port MAC preemption mode.
- enum `netc_tc_sdu_type_t` {

kNETC_PDU = 0U,

kNETC_MPDU,

kNETC_MSDU }
- Type of PDU/SDU (Protocol/Service Data Unit).
- enum `netc_port_sg_ogc_mode_t` {

kNETC_SGCheckSFD = 0U,

kNETC_SGCheckEntire }

Defines the Port's Stream Gate Open Gate Check mode.

29.5.7.2 Data Structure Documentation

29.5.7.2.1 struct `netc_port_qos_mode_t`

Data Fields

- uint8_t `qosVlanMap`: 4

Transmit QoS to VLAN PCP Mapping Profile index, only active on switch port.
- uint8_t `vlanQosMap`: 4

Receive VLAN PCP/DE to QoS Mapping Profile index, only active on switch port.
- uint8_t `defaultIpv`: 3

Port default IPV.
- uint8_t `defaultDr`: 2

Port default DR.
- bool `enVlanInfo`: 1

Enable use VLAN info to determine IPV and DR (base on VLANIPVMPaR0/1 and VLANDRMPaR)
- bool `vlanTagSelect`: 1

True: Outer VLAN, False: Inner VLAN.

Field Documentation

(1) **bool netc_port_qos_mode_t::vlanTagSelect**

Active when enVlanInfo is true

29.5.7.2.2 struct netc_port_parser_config_t

Data Fields

- uint8_t **l2PayloadCount**: 5
L2 payload fields size in bytes.
- bool **enableL3Parser**: 1
Enable/Disable parser for L3.
- uint8_t **l3PayloadCount**: 5
L3 payload fields size in bytes.
- bool **enableL4Parser**: 1
Enable/Disable parser for L4.
- uint8_t **l4PayloadCount**: 5
L4 payload fields size in bytes.

29.5.7.2.3 struct netc_port_tg_config_t

Data Fields

- uint16_t **advOffset**
Advance time offset in ns.
- uint32_t **holdSkew**
Hold-Skew in ns, not effective on ports connected to a pseudo-MAC.

Field Documentation

(1) **uint16_t netc_port_tg_config_t::advOffset**

29.5.7.2.4 struct netc_port_tc_cbs_config_t

Note

The 802.1Qav bandwidth availability parameters is calculated as follows:

- idleSlope (bits) = portTxRate * bwWeight / 100
- sendSlope (bits) = portTxRate * (100 - bwWeight) / 100
- lowCredit (bits) = tcMaxFrameSize * (100 - bwWeight) / 100
- hiCredit (bits) calculation formula depends on the traffic class, Please refer to the Reference manual.
- hiCredit (credits) = (enetClockFrequency / portTxRate) * 100 * hiCredit (bits)

Data Fields

- `uint8_t bwWeight`
Percentage units of the port transmit rate and the credit-based shaper (range from 0 ~ 100), the sum of all traffic class credit-based shaper's bandwidth cannot exceed 100.
- `uint32_t hiCredit`
*The maximum allowed accumulation of credits when conflicting transfers occur, in credit units ((enet-ClockFrequency / portTxRate) * 100)*

29.5.7.2.5 struct netc_port_tx_tc_config_t**Data Fields**

- `bool enPreemption: 1`
Frames from traffic class are transmitted on the preemptable MAC, not supported on internal port (ENETC 1 port and Switch port 4)
- `bool enTcGate: 1`
Enable the traffic class gate when no gate control list is operational, or when time gate scheduling is disabled.
- `bool enableTsd: 1`
Enable Time Specific Departure traffic class, only applicable to ENETC.
- `bool enableCbs: 1`
Enable Credit based shaper for traffic class.
- `netc_port_tc_cbs_config_t cbsCfg`
Configure transmit traffic class credit based shaper (PTC0CBSR0/PTC0CBSR1) if `enableCbs` set to true

Field Documentation(1) `bool netc_port_tx_tc_config_t::enTcGate`**29.5.7.2.6 struct netc_port_discard_statistic_t****Data Fields**

- `uint32_t count`
Count of discarded frames.
- `uint32_t reason0`
Discard Reason.
- `uint32_t reason1`
Discard Reason.

Field Documentation(1) `uint32_t netc_port_discard_statistic_t::count`

PRXDCR, PTXDCR or BPDCR.

(2) **uint32_t netc_port_discard_statistic_t::reason0**

Find bit detail from PT/RXDCRR0 or BPDCRR0.

(3) **uint32_t netc_port_discard_statistic_t::reason1**

Find bit detail from PT/RXDCRR1 or BPDCRR1.

29.5.7.2.7 struct netc_port_vlan_classify_config_t**Data Fields**

- **uint8_t innerMask:** 4
Bitmap identifying which TPIDs are acceptable as Inner VLAN tag.
- **uint8_t outerMask:** 4
Bitmap identifying which TPIDs are acceptable as Outer VLAN tag.

Field Documentation(1) **uint8_t netc_port_vlan_classify_config_t::innerMask**

See PTAR

(2) **uint8_t netc_port_vlan_classify_config_t::outerMask**

See PTAR

29.5.7.2.8 struct netc_port_qos_classify_config_t**Data Fields**

- **uint8_t vlanQosMap:** 4
Receive VLAN PCP/DE to QoS Mapping Profile index.
- **uint8_t defaultIpv:** 3
Port default IPV.
- **uint8_t defaultDr:** 2
Port default DR.
- **bool enVlanInfo:** 1
Enable use VLAN info to determine IPV and DR ,base on VLAN to IPV map (VLANIPVMPaR0/1) and VLAN to DR map (VLANDRMPaR)
- **bool vlanTagSelect:** 1
True: Use received Outer VLAN, False: Use received Innner VLAN.

Field Documentation(1) **bool netc_port_qos_classify_config_t::vlanTagSelect**

Active when enVlanInfo is true

29.5.7.2.9 struct netc_port_ipf_config_t

Data Fields

- bool `enL2Dos`: 1
Enable port L2 Ethernet DoS Protection.
- bool `enL3Dos`: 1
Enable port L3 IP DoS Protection.
- bool `enIPFTable`: 1
Enable port IPF lookup.

29.5.7.2.10 struct netc_port_psfp_isi_config

Port ingress stream identification config

Note

The first stream identification find IS_EID has higher precedence value than the second, and the priority of the IS_EID found by the IPF is specified by the IPF entry RRR bit. The possible orderings are as follows

- RRR = 00b : IPF > enKC0 > enKC1 > defaultISEID
- RRR = 01b : enKC0 > IPF > enKC1 > defaultISEID
- RRR = 10b : enKC0 > enKC1 > IPF > defaultISEID

Data Fields

- uint16_t `defaultISEID`
Default Ingress Stream Entry ID, has lower precedence value than ISI entry and IPF entry defined IS_EID.
- bool `enKC1`: 1
Enable do the second stream identification with key construction rule 1 or rule 3.
- bool `enKC0`: 1
Enable do the first stream identification with key construction rule 0 or rule 2.
- bool `kcPair`: 1
Indicates which Key Construction pair to use for this port, false - user pair0.

Field Documentation

(1) uint16_t netc_port_psfp_isi_config::defaultISEID

0xFFFF means NULL

(2) bool netc_port_psfp_isi_config::kcPair

true - use pair1 only applicable for Switch

29.5.7.2.11 struct netc_port_common_t

Data Fields

- `netc_port_vlan_classify_config_t acceptTpid`
Port acceptable VLAN tpid config.
- `netc_port_ts_select_t rxTsSelect`
Eth MAC Rx or pseudo MAC Tx timestamp clock source.
- `uint16_t pSpeed`
*Transmit Port Speed = 10Mbps * (pSpeed+1), Used by ETS, Qbu and to determine if cut-through is permissible.*
- `uint8_t rxMacsecBco`
Port receive MACSec byte count overhead which due to MACSec encapsulation.
- `uint8_t rxP pduBco`
Port receive PPDU Byte count overhead which includes IPG, SFD and Preamble.
- `uint8_t txMacsecBco`
Port transmit MACSec byte count overhead which due to MACSec encapsulation.
- `uint8_t txP pduBco`
Port transmit PPDU Byte count overhead which includes IPG, SFD and Preamble.
- `netc_port_sg_ogc_mode_t ogcMode: 1`
Stream Gate Open Gate Check mode, 0b is check whether SFD is within the open gate interval, 1b is check whether the entire frame is within the open gate interval
- `uint32_t pDelay: 24`
Link propagation delay in ns.
- `uint8_t macAddr [6]`
Port MAC address, used for Switch egress frame modification action or ENETC SI0 primary MAC address.
- `netc_port_qos_classify_config_t qosMode`
Port Rx Qos Classification config.
- `netc_port_ipf_config_t ipfCfg`
Port ingress port filter configuration.
- `netc_port_tg_config_t timeGate`
Port Tx time gate config.
- `netc_port_parser_config_t parser`
Port Rx Parser config.

Field Documentation

(1) `netc_port_vlan_classify_config_t netc_port_common_t::acceptTpid`

29.5.7.3 Enumeration Type Documentation

29.5.7.3.1 enum netc_port_tgsl_status_t

Enumerator

`kNETC_OperListActive` Port operational gate control list is active.

`kNETC_AdminListPending` Administrative gate control list is pending (configured but not installed yet).

29.5.7.3.2 enum netc_port_discard_tpye_t

Enumerator

kNETC_RxDiscard Discarded frames in the receive port datapath processing pipeline.

kNETC_TxDiscard Discarded frames in the egress datapath processing pipeline, only for switch.

kNETC_BridgeDiscard Discarded frames in the bridge forwarding processing function, only for switch.

29.5.7.3.3 enum netc_port_tpidlist_t

Enumerator

kNETC_OuterStanCvlan Accept outer Standard C-VLAN 0x8100.

kNETC_OuterStanSvlan Accept outer Standard S-VLAN 0x88A8.

kNETC_OuterCustomVlan1 Accept outer Custom VLAN as defined by CVLANR1[ETYPE].

kNETC_OuterCustomVlan2 Accept outer Custom VLAN as defined by CVLANR2[ETYPE].

kNETC_InnerStanCvlan Accept inner Standard C-VLAN 0x8100.

kNETC_InnerStanSvlan Accept inner Standard S-VLAN 0x88A8.

kNETC_InnerCustomVlan1 Accept inner Custom VLAN as defined by CVLANR1[ETYPE].

kNETC_InnerCustomVlan2 Accept inner Custom VLAN as defined by CVLANR2[ETYPE].

29.5.7.3.4 enum netc_port_ts_select_t

Enumerator

kNETC_SyncTime Synchronized time.

kNETC_FreeRunningTime Free running time.

29.5.7.3.5 enum netc_hw_preemption_mode_t

Enumerator

kNETC_PreemptDisable Frame preemption is not enabled.

kNETC_PreemptOn64B Frame preemption is enabled, but transmit only preempts frames on 64B boundaries.

kNETC_PreemptOn4B Frame preemption is enabled, but transmit only preempts frames on 4B boundaries.

29.5.7.3.6 enum netc_tc_sdu_type_t

Note

Overhead values which adding to the transmitted frame of Length are specified by Port SDU config as follows:

- PPDU = add rxPpduBco/txPpduBco + rxMacsecBco/txMacsecBco bytes
- MPDU = add rxMacsecBco/txMacsecBco bytes
- MSDU = minus 16B (12B MAC Header + 4B FCS)

Enumerator

kNETC_PDU Physical Layer PDU, Preamble, IFG, SFD along with MPDU. Not supported if cut-through frames are expected

kNETC_MPDU MAC PDU, MAC Header, MSDU and FCS.

kNETC_MSDU MAC SDU, MPDU minus 12B MAC Header and 4B FCS. Not supported if cut-through frames are expected

29.5.7.3.7 enum netc_port_sg_ogc_mode_t

Enumerator

kNETC_SGCheckSFD Check whether frame SFD is within the open gate interval.

kNETC_SGCheckEntire Check whether the entire frame is within the open gate interval.

29.5.7.4 Hardware Port Tx

Group for the Port Tx functionalities

29.5.7.5 Hardware Port Rx

Group for the Port Rx functionalities

29.5.8 Hardware Port MAC

29.5.8.1 Overview

Group for the Port Mac functionalities

Data Structures

- struct `netc_port_phy_mac_preemption_status_t`
Port MAC preemption Status. [More...](#)
- struct `netc_port_phy_mac_traffic_statistic_t`
Ethernet MAC physical port traffic (Tx/Rx) statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups. [More...](#)
- struct `netc_port_phy_mac_discard_statistic_t`
Ethernet MAC physical port frame discard/errors status statistics counters, when enable frame preemption, one physical MAC will be divided into a pMAC and a eMAC and statistics counters will also have two groups. [More...](#)
- struct `netc_port_phy_mac_preemption_statistic_t`
Ethernet physical MAC port preemption (Tx/Rx) related statistics counters. [More...](#)
- struct `netc_port_pseudo_mac_traffic_statistic_t`
Ethernet pseudo MAC port traffic (Tx/Rx) statistics counters. [More...](#)

Enumerations

- enum `netc_port_phy_mac_type_t` {
`kNETC_ExpressMAC` = 0U,
`kNETC_PreemptableMAC` }
Defines the Ethernet MAC physical port type.
- enum `netc_port_preemption_verify_status_t` {
`kNETC_VerifyDisable` = 0U,
`kNETC_VerifyInProgress` = 2U,
`kNETC_VerifySuccess`,
`kNETC_VerifyFail`,
`kNETC_VerifyUndefined` }
Defines the state of the mac merge sublayer with respect to verification as defined in IEEE Std 802.3br-2016.

29.5.8.2 Data Structure Documentation

29.5.8.2.1 struct `netc_port_phy_mac_preemption_status_t`

Data Fields

- bool `mergeActive`
Transmit preemption is active or not.

- `netc_port_preemption_verify_status_t verifyStatus`
Transmit preemption is active or not.

29.5.8.2.2 struct netc_port_phy_mac_traffic_statistic_t

Data Fields

- `uint64_t totalOctet`
Count of MAC received/transmitted good/error Ethernet octets.
- `uint64_t validOctet`
Count of MAC received/transmitted good Ethernet octets.
- `uint64_t pauseFrame`
Count of MAC received/transmitted valid PAUSE frames.
- `uint64_t validFrame`
Count of MAC received/transmitted valid frames.
- `uint64_t vlanFrame`
Count of MAC received/transmitted valid VLAN tagged frames.
- `uint64_t unicastFrame`
Count of MAC received/transmitted valid unicast frames.
- `uint64_t multicastFrame`
Count of MAC received/transmitted valid multicast frames.
- `uint64_t broadcastFrame`
Count of MAC received/transmitted valid broadcast frames.
- `uint64_t totalPacket`
Count of MAC received/transmitted good/error packets.
- `uint64_t rxMinPacket`
Count of MAC received min to 63-octet packets.
- `uint64_t total64BPacket`
Count of MAC received/transmitted 64 octet packets.
- `uint64_t total65To127BPacket`
Count of MAC received/transmitted 65 to 127 octet packets.
- `uint64_t total128To255BPacket`
Count of MAC received/transmitted 128 to 255 octet packets.
- `uint64_t total256To511BPacket`
Count of MAC received/transmitted 256 to 511 octet packets.
- `uint64_t total512To1023BPacket`
Count of MAC received/transmitted 512 to 1023 octet packets.
- `uint64_t total1024To1522BPacket`
Count of MAC received/transmitted 1024 to 1522 octet packets.
- `uint64_t total1523ToMaxBPacket`
Count of MAC received/transmitted 1523 to Max octet packets.
- `uint64_t controlPacket`
Count of MAC received/transmitted control packets.

Field Documentation

- (1) `uint64_t netc_port_phy_mac_traffic_statistic_t::totalOctet`
- (2) `uint64_t netc_port_phy_mac_traffic_statistic_t::validOctet`
- (3) `uint64_t netc_port_phy_mac_traffic_statistic_t::pauseFrame`
- (4) `uint64_t netc_port_phy_mac_traffic_statistic_t::validFrame`
- (5) `uint64_t netc_port_phy_mac_traffic_statistic_t::vlanFrame`
- (6) `uint64_t netc_port_phy_mac_traffic_statistic_t::unicastFrame`
- (7) `uint64_t netc_port_phy_mac_traffic_statistic_t::multicastFrame`
- (8) `uint64_t netc_port_phy_mac_traffic_statistic_t::broadcastFrame`
- (9) `uint64_t netc_port_phy_mac_traffic_statistic_t::totalPacket`
- (10) `uint64_t netc_port_phy_mac_traffic_statistic_t::rxMinPacket`
- (11) `uint64_t netc_port_phy_mac_traffic_statistic_t::total64BPacket`
- (12) `uint64_t netc_port_phy_mac_traffic_statistic_t::total65To127BPacket`
- (13) `uint64_t netc_port_phy_mac_traffic_statistic_t::total128To255BPacket`
- (14) `uint64_t netc_port_phy_mac_traffic_statistic_t::total256To511BPacket`
- (15) `uint64_t netc_port_phy_mac_traffic_statistic_t::total511To1023BPacket`
- (16) `uint64_t netc_port_phy_mac_traffic_statistic_t::total1024To1522BPacket`
- (17) `uint64_t netc_port_phy_mac_traffic_statistic_t::total1523ToMaxBPacket`
- (18) `uint64_t netc_port_phy_mac_traffic_statistic_t::controlPacket`

29.5.8.2.3 struct netc_port_phy_mac_discard_statistic_t

Data Fields

- `uint64_t rxError`
Count of MAC received error frames.
- `uint64_t rxUndersized`
Count of MAC received undersized frames.
- `uint64_t rxOversized`
Count of MAC received oversized frames.
- `uint64_t rxErrorFCS`
Count of MAC received check sequence (FCS) error frames.
- `uint64_t rxFragment`

- `uint64_t rxJabber`
Count of MAC frames which is shorter than the MIN length and received with a wrong FCS/CRC.
- `uint64_t rxDiscard`
Count of MAC frames which is larger than the MAX length and received with a wrong FCS/CRC.
- `uint64_t rxDiscardNoTruncated`
Count of MAC drops frame.
- `uint64_t txErrorFCS`
Count of MAC non-truncated drops frame.
- `uint64_t txUndersized`
Count of MAC transmitted bad FCS frames.
- `uint64_t rxOversized`
Count of MAC transmitted less than 64B with good FCS frames.

Field Documentation

- (1) `uint64_t netc_port_phy_mac_discard_statistic_t::rxError`
- (2) `uint64_t netc_port_phy_mac_discard_statistic_t::rxUndersized`
- (3) `uint64_t netc_port_phy_mac_discard_statistic_t::rxOversized`
- (4) `uint64_t netc_port_phy_mac_discard_statistic_t::rxErrorFCS`
- (5) `uint64_t netc_port_phy_mac_discard_statistic_t::rxFragment`
- (6) `uint64_t netc_port_phy_mac_discard_statistic_t::rxJabber`
- (7) `uint64_t netc_port_phy_mac_discard_statistic_t::rxDiscard`
- (8) `uint64_t netc_port_phy_mac_discard_statistic_t::rxDiscardNoTruncated`
- (9) `uint64_t netc_port_phy_mac_discard_statistic_t::txErrorFCS`
- (10) `uint64_t netc_port_phy_mac_discard_statistic_t::txUndersized`

29.5.8.2.4 struct netc_port_phy_mac_preemption_statistic_t

Data Fields

- `uint32_t rxReassembledFrame`
Count of MAC frames that were successfully reassembled and delivered to the MAC.
- `uint32_t rxReassembledError`
Count of MAC frames with reassembly errors.
- `uint32_t rxMPacket`
Count of the number of additional mPackets received due to preemption.
- `uint32_t rxSMDError`
Count of received MAC frames / MAC frame fragments rejected due to unknown SMD.
- `uint32_t txPreemptionReq`
Count of the number of tx preemption HOLD requests.
- `uint32_t txMPacket`
Count of the number of additional mPackets transmitted due to preemption.

Field Documentation

- (1) `uint32_t netc_port_phy_mac_preemption_statistic_t::rxReassembledFrame`
- (2) `uint32_t netc_port_phy_mac_preemption_statistic_t::rxReassembledError`
- (3) `uint32_t netc_port_phy_mac_preemption_statistic_t::rxMPacket`
- (4) `uint32_t netc_port_phy_mac_preemption_statistic_t::rxSMDError`
- (5) `uint32_t netc_port_phy_mac_preemption_statistic_t::txPreemptionReq`
- (6) `uint32_t netc_port_phy_mac_preemption_statistic_t::txMPacket`

29.5.8.2.5 struct netc_port_pseudo_mac_traffic_statistic_t**Data Fields**

- `uint64_t totalOctet`
Count of MAC received/transmitted octets.
- `uint64_t unicastFrame`
Count of MAC received/transmitted unicast frames.
- `uint64_t multicastFrame`
Count of MAC received/transmitted multicast frames.
- `uint64_t broadcastFrame`
Count of MAC received/transmitted broadcast frames .

Field Documentation

- (1) `uint64_t netc_port_pseudo_mac_traffic_statistic_t::totalOctet`
- (2) `uint64_t netc_port_pseudo_mac_traffic_statistic_t::unicastFrame`
- (3) `uint64_t netc_port_pseudo_mac_traffic_statistic_t::multicastFrame`
- (4) `uint64_t netc_port_pseudo_mac_traffic_statistic_t::boradcastFrame`

29.5.8.3 Enumeration Type Documentation**29.5.8.3.1 enum netc_port_phy_mac_type_t**

Enumerator

kNETC_ExpressMAC The MAC which handles express traffic when frame preemption is enabled or handles all traffic when frame preemption is disabled.

kNETC_PreemptableMAC The MAC which handles preemptive traffic when frame preemption is enabled.

29.5.8.3.2 enum netc_port_preemption_verify_status_t

Enumerator

kNETC_VerifyDisable Verification is disabled.

kNETC_VerifyInProgress Verification is in progress.

kNETC_VerifySuccess Verification was successful.

kNETC_VerifyFailed Verification failed.

kNETC_VerifyUndefined Verification is in an undefined state.

29.5.9 Hardware Switch

29.5.9.1 Overview

Group for switch specific hardware API and data structure.

Data Structures

- struct `netc_swt_imr_config_t`
Switch Ingress mirror destination config. [More...](#)
- struct `netc_swt_port_bridge_config_t`
Switch port bridge configuration. [More...](#)
- struct `netc_swt_port_fm_config_t`
Switch Port level Frame Modification configuration (PPCPDEIMR and PQOSMR[QVMP]) [More...](#)
- struct `netc_swt_default_vlan_filter_t`
Switch VLAN filter hash table default entry configuration, which determines the default entry when not found in VLAN filter lookup. [More...](#)
- struct `netc_swt_bridge_config_t`
Bridge config. [More...](#)
- struct `netc_swt_psfp_config_t`
Switch PSFP configuration. [More...](#)
- struct `netc_swt_qos_classify_config_t`
Switch Qos Classification configuration (include two profiles) [More...](#)
- struct `netc_swt_qos_to_vlan_config_t`
Switch QoS to PCP mapping and PCP to PCP mapping configuration when egress packet modification the VLAN tag. [More...](#)
- struct `netc_switch_inuse_fdb_statistic_t`
Switch static/dynamic FDB entries in-use statistic. [More...](#)

Enumerations

- enum `netc_swt_port_bitmap_t` {

`kNETC_SWTPort0Bit` = 0x1U,

`kNETC_SWTPort1Bit` = 0x2U,

`kNETC_SWTPort2Bit` = 0x4U,

`kNETC_SWTPort3Bit` = 0x8U,

`kNETC_SWTPort4Bit` = 0x10U
 }

The switch port bitmap.
- enum `netc_swt_imr_dest_port_t` {

`kNETC_SWTPort0` = 0U,

`kNETC_SWTPort1` = 1U,

`kNETC_SWTPort2` = 2U,

`kNETC_SWTPort3` = 3U,

`kNETC_SWTPort4` = 4U,

`kNETC_SWTPort5` = 5U
 }

The switch ingress mirror destination port.

- enum `netc_swt_mac_forward_mode_t` {

 `kNETC_NoFDBLookUp` = 1U,

 `kNETC_FDBLookUpWithFlood` = 2U,

 `kNETC_FDBLookUpWithDiscard` = 3U }

 The switch MAC forwarding options.
- enum `netc_swt_mac_learn_mode_t` {

 `kNETC_DisableMACLearn` = 1U,

 `kNETC_HardwareMACLearn`,

 `kNETC_SeSoftwareMACLearn`,

 `kNETC_UnseSoftwareMACLearn`,

 `kNETC_DisableMACLearnWithSMAC` }

 The switch MAC learning options.
- enum `netc_swt_port_tx_vlan_act_t` {

 `kNETC_NoTxVlanModify` = 0U,

 `kNETC_TxDelOuterVlan`,

 `kNETC_TxReplOuterVlanVid` }

 Switch transmit Bridge Port VLAN Tag Action.
- enum `netc_swt_port_stg_mode_t` {

 `kNETC_DiscardFrame` = 0U,

 `kNETC_LearnWithoutForward`,

 `kNETC_ForwardFrame` }

 Switch port spanning tree group work mode.

29.5.9.2 Data Structure Documentation

29.5.9.2.1 struct `netc_swt_imr_config_t`

Data Fields

- bool `enMirror`

 Enable ingress mirroring.
- `netc_swt_imr_dest_port_t destPort`

 Port where ingress mirrored frames are sent.
- `uint8_t dr`

 Mirrored packet's DR (drop resilience)
- `uint8_t ipv`

 Mirrored packet's IPV (internal priority value)
- `uint8_t efmLengthChange`

 Egress Frame Modification Frame Length change in 2s complement notation, Valid if efmEntryID is not null.
- `uint16_t efmEntryID`

 Egress Frame Modification Entry Id, note 0xFFFF is a Null Frame Modification Entry, Only applicable if destPort != kNETC_SWTPort.

29.5.9.2.2 struct netc_swt_port_bridge_config_t

Data Fields

- **netc_swt_port_tx_vlan_act_t txVlanAction:** 2
Only applies for the frame outer VLAN tag's VID is equal to the port default VID.
- **bool isRxVlanAware:** 1
Receive VLAN Aware Mode.
- **bool acceptUntag:** 1
Accept untagged frame.
- **bool acceptPriorityTag:** 1
Accept priority tagged frame (VID = 0)
- **bool acceptSingleTag:** 1
Accept single tagged frame.
- **bool acceptDoubleTag:** 1
Accept double tagged frame (outer and inner)
- **bool enSrcPortPrun:** 1
Enable/Disable received frame be transmitted to same port it was received.
- **bool enMacStationMove:** 1
Enable/Disable received frame which ingress port not match the FDB entry Destination Port Bitmap.
- **bool enBcastStormCtrl:** 1
Enable/Disable Storm control for broadcast frames.
- **bool enMcastStormCtrl:** 1
Enable/Disable Storm control for multicast frames.
- **bool enUnMcastStormCtrl:** 1
Enable/Disable Storm control for unknown multicast frames.
- **bool enUnUcastStormCtrl:** 1
Enable/Disable Storm control for unknown unicast frames.
- **uint32_t bcastRpEntryID**
Broadcast rate policer entry ID.
- **uint32_t mcastEntryID**
Known multicast rate policer entry ID.
- **uint32_t unMcastRpEntryID**
Unknown multicast policer entry ID.
- **uint32_t unUcastRpEntryID**
Unknown unicast rate policer entry ID.
- **uint16_t maxDynaFDBEntry**
The maximum number of dynamic entries in the FDB table, 0 means no limit.
- **uint32_t vid:** 12
Vlan Identifier.
- **uint32_t dei:** 1
Drop eligible indicator.
- **uint32_t pcp:** 3
Priority code point.
- **uint32_t tpid:** 1
Tag protocol identifier, 0 = Standard C-VLAN 0x8100, 1 = Standard S-VLAN 0x88A8.

Field Documentation

- (1) `uint32_t netc_swt_port_bridge_config_t::vid`
- (2) `uint32_t netc_swt_port_bridge_config_t::pcp`
- (3) `uint32_t netc_swt_port_bridge_config_t::tpid`
- (4) `uint32_t netc_swt_port_bridge_config_t::bcastRpEntryID`

Valid if enBroadStormCtrl = true

- (5) `uint32_t netc_swt_port_bridge_config_t::mcastEntryID`

Valid if enBroadStormCtrl = true

- (6) `uint32_t netc_swt_port_bridge_config_t::unMcastRpEntryID`

Valid if enUnMultiStormCtrl = true

- (7) `uint32_t netc_swt_port_bridge_config_t::unUcastRpEntryID`

Valid if enUnUniStormCtrl = true

29.5.9.2.3 struct netc_swt_port_fm_config_t**Data Fields**

- bool `ignoreFMMiscfg`: 1
Enable/Disable ignore the Frame Modification Misconfiguration Action.
- bool `enEgressPcpMap`: 1
Enable egress frame modification of outer VLAN tag's PCP value is mapped to a new value based on egressPcpMap, used for Frame Modification VLAN Outer PCP action.
- bool `enIngressPcpMap`: 1
Enable ingress frame modification of outer VLAN tag's PCP value is mapped to a new value based on egressPcpMap, used for Frame Modification VLAN Outer PCP action.
- bool `enUpdateVlanDei`: 1
Enable update DR value in the outer VLAN based on DEnDEI field, used for egress Frame Modification Outer DEI action.
- uint8_t `drToDeiMap`: 4
Mapping of internal QoS's DR value n to VLAN DEI, The 4 bits correspond to the DR3 ~ DR0, and 1 means DRn mapping to DEI 1, 0 means DRn mapping to DEI 0.
- uint8_t `egressPcpMap`: 4
Egress PCP to PCP Mapping Profile instance, active when enEgressPcpMap is true.
- uint8_t `ingressPcpMap`: 4
Ingress PCP to PCP Mapping Profile instance, active when enIngressPcpMap is true.
- uint8_t `qosVlanMap`: 4
Transmit QoS to VLAN PCP Mapping Profile index, used for egress Frame Modification VLAN Add-/Replace Action.

29.5.9.2.4 struct netc_swt_default_vlan_filter_t

Data Fields

- bool **enIPMFlood**
Enable IP Multicast Flooding.
- bool **enIPMFilter**
Enable IP Multicast Filtering.
- uint8_t **stgID**
Spanning Tree Group Member ID, range in 0 ~ 15.
- uint8_t **portMembership**
The bit 0 ~ 4 correspond to the 5 ports, When bit set (0b1), means the port is a member of this VLAN.
- bool **enUseFilterID**
Enable use the specified filterID as FID, otherwise will use the frame VID.
- uint16_t **filterID**
Used as a key value to do FDB table and the L2 IPV4 Multicast Filter table lookup.
- **netc_swt_mac_forward_mode_t mfo**
MAC forwarding options.
- **netc_swt_mac_learn_mode_t mlo**
MAC learning options.
- uint16_t **baseETEID**
Base Egress Treatment Entry ID.
- uint8_t **etaPortBitmap**
Egress Treatment Applicability Port.

Field Documentation

(1) **uint8_t netc_swt_default_vlan_filter_t::portMembership**

Port membership is used for source/destination pruning

(2) **uint16_t netc_swt_default_vlan_filter_t::filterID**

Valid if enUseFilterID is true

(3) **uint8_t netc_swt_default_vlan_filter_t::etaPortBitmap**

Valid if baseETEID is not null.

29.5.9.2.5 struct netc_swt_bridge_config_t

Data Fields

- **netc_swt_default_vlan_filter_t dVFCfg**
Default VLAN filter entry configuration when not found in VLAN filter lookup.

29.5.9.2.6 struct netc_swt_psfp_config_t

Data Fields

- `netc_isi_kc_rule_t kcRule [4]`

Key construction rules.

29.5.9.2.7 struct netc_swt_qos_classify_config_t

29.5.9.2.8 struct netc_swt_qos_to_vlan_config_t

29.5.9.2.9 struct netc_switch_inuse_fdb_statistic_t

Data Fields

- `uint16_t camEntries`

Number of FDB entries in-use in the CAM.

- `uint16_t staticEntries`

Number of static FDB entries in-use (both hash-based and CAM-based entries).

- `uint16_t dynamicEntries`

Number of dynamic FDB entries in-use (hash-based and CAM-based entries).

- `uint16_t dynamicEntriesHWM`

High water mark of dynamic entries in-use in the FDB table.

Field Documentation

- (1) `uint16_t netc_switch_inuse_fdb_statistic_t::camEntries`
- (2) `uint16_t netc_switch_inuse_fdb_statistic_t::staticEntries`
- (3) `uint16_t netc_switch_inuse_fdb_statistic_t::dynamicEntries`
- (4) `uint16_t netc_switch_inuse_fdb_statistic_t::dynamicEntriesHWM`

29.5.9.3 Enumeration Type Documentation

29.5.9.3.1 enum netc_swt_port_bitmap_t

Enumerator

- `kNETC_SWTPort0Bit` Switch port0 bitmap.
- `kNETC_SWTPort1Bit` Switch port1 bitmap.
- `kNETC_SWTPort2Bit` Switch port2 bitmap.
- `kNETC_SWTPort3Bit` Switch port3 bitmap.
- `kNETC_SWTPort4Bit` Switch port4 (internal port) bitmap.

29.5.9.3.2 enum netc_swt_imr_dest_port_t

Enumerator

- kNETC_SWTPort0* Switch port0.
- kNETC_SWTPort1* Switch port1.
- kNETC_SWTPort2* Switch port2.
- kNETC_SWTPort3* Switch port3.
- kNETC_SWTPort4* Switch port4.
- kNETC_SWTMPort* Switch management port.

29.5.9.3.3 enum netc_swt_mac_forward_mode_t

Enumerator

- kNETC_NoFDBLookUp* No FDB lookup is performed, the frame is flooded.
- kNETC_FDBLookUpWithFlood* FDB lookup is performed, and if there is no match, the frame is flooded to the port bitmap in VLAN filter entry.
- kNETC_FDBLookUpWithDiscard* FDB lookup is performed, and if there is no match, the frame is discarded.

29.5.9.3.4 enum netc_swt_mac_learn_mode_t

Enumerator

- kNETC_DisableMACLearn* Disable MAC learning. SMAC FDB lookup is by-passed.
- kNETC_HardwareMACLearn* Hardware MAC learning is enabled.
- kNETC_SeSoftwareMACLearn* Software MAC learning secure. FDB lookup based on FID and S-MAC is performed and if an entry is not found, the frame is redirected to the switch management port.
- kNETC_UnseSoftwareMACLearn* Software MAC learning unsecure. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is copied to the switch management port.
- kNETC_DisableMACLearnWithSMAC* Disable MAC learning with SMAC validation. FDB lookup based on FID and SMAC is performed and if an entry is not found, the frame is discarded.

29.5.9.3.5 enum netc_swt_port_tx_vlan_act_t

Enumerator

- kNETC_NoTxVlanModify* No egress VLAN modification performed.
- kNETC_TxDelOuterVlan* Delete outer VLAN tag.
- kNETC_TxReplOuterVlanVid* Replace outer VLAN tag's VID with 0; frame to be transmitted as a priority tag frame.

29.5.9.3.6 enum netc_swt_port_stg_mode_t

Enumerator

kNETC_DiscardFrame Tx or RX Frames on this port with current spanning tree group ID will be discarded.

kNETC_LearnWithoutForward RX Frames on this port with current spanning tree group ID will do Learn SMAC, but do not forward, Tx Frame will be discarded.

kNETC_ForwardFrame RX Frames on this port with current spanning tree group ID will do both MAC learning and forwarding, , Tx Frame will be forwarded.

29.5.10 Hardware Common Functions

29.5.10.1 Overview

Group for API/Data structure of common usage.

Functions

- `uint16_t NETC_SIGetVsiIndex (netc_vsi_number_t vsi)`
Get the VSI index.
- `static void NETC_IPFInit (NETC_SW_ENETC_Type *base, const netc_ipf_config_t *config)`
Set layer2/3 Dos configuration.
- `void NETC_PSFPKcProfileInit (NETC_SW_ENETC_Type *base, const netc_isi_kc_rule_t *rule, bool enKcPair1)`
Initialize the Ingress Stream Identification Key construction rule profiles.
- `void NETC_RxVlanCInit (NETC_SW_ENETC_Type *base, const netc_vlan_classify_config_t *config, bool enRtag)`
Initialize the customer vlan type.
- `void NETC_RxQosCInit (NETC_SW_ENETC_Type *base, const netc_qos_classify_profile_t *profile, bool enProfile1)`
Initialize the ingress QoS classification.

29.5.10.2 Function Documentation

29.5.10.2.1 `uint16_t NETC_SIGetVsiIndex (netc_vsi_number_t vsi)`

Parameters

<code>vsi</code>	The VSI number.
------------------	-----------------

29.5.10.2.2 `static void NETC_IPFInit (NETC_SW_ENETC_Type * base, const netc_ipf_config_t * config) [inline], [static]`

Parameters

<code>base</code>	
<code>config</code>	

29.5.10.2.3 `void NETC_PSFPKcProfileInit (NETC_SW_ENETC_Type * base, const netc_isi_kc_rule_t * rule, bool enKcPair1)`

Parameters

<i>base</i>	
<i>rule</i>	
<i>enKcPair1</i>	

Returns

void

29.5.10.2.4 void NETC_RxVlanCInit (NETC_SW_ENETC_Type * *base*, const netc_vlan_classify_config_t * *config*, bool *enRtag*)

Parameters

<i>base</i>	
<i>config</i>	
<i>enRtag</i>	

Returns

void

29.5.10.2.5 void NETC_RxQosCInit (NETC_SW_ENETC_Type * *base*, const netc_qos_classify_profile_t * *profile*, bool *enProfile1*)

Parameters

<i>base</i>	
<i>profile</i>	
<i>enProfile1</i>	

29.5.11 Hardware Table Access Functions

29.5.11.1 Overview

Group for API/Data structure of Tables. It supports NTMP 1.0 and NTMP 2.0 but it is transparent to the invoker.

Data Structures

- union `netc_cmd_bd_t`
The Switch/SI command BD data structure. [More...](#)
- struct `netc_cmd_bdr_config_t`
Configuration for the Switch/SI command BD Ring Configuration. [More...](#)
- struct `netc_cmd_bdr_t`
The Switch/SI command BD ring handle data structure. [More...](#)
- struct `netc_tb_common_header_t`
Table request data buffer common header. [More...](#)
- struct `netc_tb_ipf_keye_t`
Ingress Port Filter key element. [More...](#)
- struct `netc_tb_ipf_cfge_t`
Ingress port filter config element. [More...](#)
- struct `netc_tb_ipf_stse_t`
Ingress port filter statistic element. [More...](#)
- struct `netc_tb_ipf_req_data_t`
Ingress port filter table entry config. [More...](#)
- struct `netc_tb_ipf_rsp_data_t`
Ingress port filter table response data. [More...](#)
- struct `netc_tb_ipf_data_t`
Ingress Port filter table data buffer. [More...](#)
- struct `netc_tb_ipf_config_t`
Ingress Port filter entry config. [More...](#)
- struct `netc_tb_isi_keye_t`
Stream identification table key element. [More...](#)
- struct `netc_tb_isi_cfge_t`
Stream identification table config element. [More...](#)
- struct `netc_tb_isi_req_data_t`
Stream identification table request data buffer. [More...](#)
- struct `netc_tb_isi_rsp_data_t`
Stream identification table request response data buffer. [More...](#)
- struct `netc_tb_isi_data_t`
Stream identification table data buffer. [More...](#)
- struct `netc_tb_isi_config_t`
Stream identification table entry config. [More...](#)
- struct `netc_tb_is_cfge_t`
Ingress Stream table config element. [More...](#)
- struct `netc_tb_is_req_data_t`
Ingress Stream table request data buffer. [More...](#)
- struct `netc_tb_is_rsp_data_t`
Ingress Stream table request response data buffer. [More...](#)
- struct `netc_tb_is_data_t`

- struct [netc_tb_is_config_t](#)
Ingress Stream table data buffer. [More...](#)
- struct [netc_tb_isf_keye_t](#)
Ingress Stream table entry config. [More...](#)
- struct [netc_tb_isf_cfge_t](#)
Ingress Stream Filter table key element. [More...](#)
- struct [netc_tb_isf_cfge_t](#)
Ingress Stream Filter table config element. [More...](#)
- struct [netc_tb_isf_req_data_t](#)
Ingress Stream Filter table request data buffer. [More...](#)
- struct [netc_tb_isf_rsp_data_t](#)
Ingress Stream Filter table request response data buffer. [More...](#)
- struct [netc_tb_isf_data_t](#)
Ingress Stream Filter table data buffer. [More...](#)
- struct [netc_tb_isf_config_t](#)
Ingress Stream Filter table entry config. [More...](#)
- struct [netc_tb_rp_cfge_t](#)
Rate Policer table config element. [More...](#)
- struct [netc_tb_rp_fee_t](#)
Rate Policer table Function Enable element. [More...](#)
- struct [netc_tb_rp_pse_t](#)
Rate Policer table Policer State element. [More...](#)
- struct [netc_tb_rp_stse_t](#)
Rate Policer table statistic element. [More...](#)
- struct [netc_tb_rp_req_data_t](#)
Rate Policer table request data buffer. [More...](#)
- struct [netc_tb_rp_rsp_data_t](#)
Rate Policer table request response data buffer. [More...](#)
- struct [netc_tb_rp_data_t](#)
Rate Policer table data buffer. [More...](#)
- struct [netc_tb_rp_config_t](#)
Rate Policer table entry config. [More...](#)
- struct [netc_tb_isc_stse_t](#)
Ingress Stream Count table statistic element. [More...](#)
- struct [netc_tb_isc_req_data_t](#)
Ingress Stream Count table request data buffer. [More...](#)
- struct [netc_tb_isc_rsp_data_t](#)
Ingress Stream Count table request response data buffer. [More...](#)
- struct [netc_tb_isc_data_t](#)
Ingress Stream Count table data buffer. [More...](#)
- struct [netc_tb_sgi_cfge_t](#)
Stream Gate Instance table config element. [More...](#)
- struct [netc_tb_sgi_acfge_t](#)
Stream Gate Instance table Admin Configuration element. [More...](#)
- struct [netc_tb_sgi_icfge_t](#)
Stream Gate Instance table Initial Configuration element. [More...](#)
- struct [netc_tb_sgi_sgise_t](#)
Stream Gate Instance table stream gate instance state element. [More...](#)
- struct [netc_tb_sgi_req_data_t](#)
Stream Gate Instance table request data buffer. [More...](#)
- struct [netc_tb_sgi_rsp_data_t](#)
Stream Gate Instance table request response data buffer. [More...](#)

- struct [netc_tb_sgi_data_t](#)
Stream Gate Instance table data buffer. [More...](#)
- struct [netc_tb_sgi_config_t](#)
Stream Gate Instance table entry config. [More...](#)
- struct [netc_sgcl_gate_entry_t](#)
Defines the Stream Gate Control entry structure. [More...](#)
- struct [netc_tb_sgcl_cfge_t](#)
Stream Gate Control List table config element. [More...](#)
- struct [netc_tb_sgcl_sgclse_t](#)
Stream Gate Control List table Stream Gate Control List State element. [More...](#)
- struct [netc_tb_sgcl_req_data_t](#)
Stream Gate Control List table request data buffer. [More...](#)
- struct [netc_tb_sgcl_rsp_data_t](#)
Stream Gate Control List table request response data buffer. [More...](#)
- struct [netc_tb_sgcl_data_t](#)
Stream Gate Control List table data buffer. [More...](#)
- struct [netc_tb_sgcl_gcl_t](#)
Stream Gate Control List table entry gate control list structure. [More...](#)
- struct [netc_tb_fm_cfge_t](#)
Frame Modification table config element. [More...](#)
- struct [netc_tb_fm_req_data_t](#)
Frame Modification table request data buffer. [More...](#)
- struct [netc_tb_fm_rsp_data_t](#)
Frame Modification table request response data buffer. [More...](#)
- struct [netc_tb_fm_data_t](#)
Frame Modification table data buffer. [More...](#)
- struct [netc_tb_fm_config_t](#)
Frame Modification table entry config. [More...](#)
- struct [netc_tb_fmd_req_data_t](#)
Frame Modification Data table request data buffer. [More...](#)
- struct [netc_tb_fmd_rsp_data_t](#)
Frame Modification Data table request response data buffer. [More...](#)
- struct [netc_tb_fmd_data_t](#)
Frame Modification Data table data buffer. [More...](#)
- struct [netc_tb_fmd_update_config_t](#)
Frame Modification data table entry update config. [More...](#)
- struct [netc_tb_fmd_query_buffer_t](#)
Frame Modification data table entry query data buffer. [More...](#)
- struct [netc_tb_vf_keye_t](#)
Vlan Filter table key element. [More...](#)
- struct [netc_tb_vf_cfge_t](#)
Vlan Filter table config element. [More...](#)
- struct [netc_tb_vf_search_criteria_t](#)
Vlan Filter table search criteria format. [More...](#)
- struct [netc_tb_vf_req_data_t](#)
Vlan Filter table request data buffer. [More...](#)
- struct [netc_tb_vf_rsp_data_t](#)
Vlan Filter table request response data buffer. [More...](#)
- struct [netc_tb_vf_data_t](#)
Vlan Filter table data buffer. [More...](#)
- struct [netc_tb_vf_config_t](#)

- Vlan Filter table entry config. [More...](#)
- struct `netc_tb_fdb_cfge_t`
FDB table configuration element. [More...](#)
- struct `netc_tb_fdb_acte_t`
FDB table Activity element. [More...](#)
- struct `netc_tb_fdb_search_criteria_t`
FDB table search criteria format. [More...](#)
- struct `netc_tb_fdb_req_data_t`
FDB table request data buffer. [More...](#)
- struct `netc_tb_fdb_rsp_data_t`
FDB table request response data buffer. [More...](#)
- struct `netc_tb_fdb_data_t`
FDB table data buffer. [More...](#)
- struct `netc_tb_fdb_config_t`
FDB table entry config. [More...](#)
- struct `netc_tb_l2mcf_keye_t`
L2 IPV4 Multicast Filter table key element. [More...](#)
- struct `netc_tb_l2mcf_search_criteria_t`
L2 IPV4 Multicast Filter table search criteria format. [More...](#)
- struct `netc_tb_l2mcf_req_data_t`
L2 IPV4 Multicast Filter table request data buffer. [More...](#)
- struct `netc_tb_l2mcf_rsp_data_t`
L2 IPV4 Multicast Filter table request response data buffer. [More...](#)
- struct `netc_tb_l2mcf_data_t`
L2 IPV4 Multicast Filter table data buffer. [More...](#)
- struct `netc_tb_l2mcf_config_t`
L2 IPV4 Multicast Filter table entry config. [More...](#)
- struct `netc_tb_iseqg_cfge_t`
Ingress Sequence Generation table config element. [More...](#)
- struct `netc_tb_iseqg_sgse_t`
Ingress Sequence Generation table Sequence generation state element. [More...](#)
- struct `netc_tb_iseqg_req_data_t`
Ingress Sequence Generation table request data buffer. [More...](#)
- struct `netc_tb_iseqg_rsp_data_t`
Ingress Sequence Generation table request response data buffer. [More...](#)
- struct `netc_tb_iseqg_data_t`
Ingress Sequence Generation table data buffer. [More...](#)
- struct `netc_tb_iseqg_config_t`
Ingress Sequence Generation table entry config. [More...](#)
- struct `netc_tb_eseqr_cfge_t`
Egress Sequence Recovery table config element. [More...](#)
- struct `netc_tb_eseqr_stse_t`
Egress Sequence Recovery table statistic element. [More...](#)
- struct `netc_tb_eseqr_srse_t`
Egress Sequence Recovery table sequence recovery state element. [More...](#)
- struct `netc_tb_eseqr_req_data_t`
Egress Sequence Recovery table request data buffer. [More...](#)
- struct `netc_tb_eseqr_rsp_data_t`
Egress Sequence Recovery table request response data buffer. [More...](#)
- struct `netc_tb_eseqr_data_t`
Egress Sequence Recovery table data buffer. [More...](#)

- struct `netc_tb_eseqr_config_t`
Egress Sequence Recovery table entry config. [More...](#)
- struct `netc_tgs_gate_entry_t`
Defines the Time Gate Scheduling gate control entry structure. [More...](#)
- struct `netc_tb_tgs_cfge_t`
Time Gate Scheduling table config element. [More...](#)
- struct `netc_tb_tgs_olse_t`
Time Gate Scheduling table statistic element. [More...](#)
- struct `netc_tb_tgs_req_data_t`
Time Gate Scheduling table request data buffer. [More...](#)
- struct `netc_tb_tgs_rsp_data_t`
Time Gate Scheduling table request response data buffer. [More...](#)
- struct `netc_tb_tgs_data_t`
Time Gate Scheduling table data buffer, set with max size. [More...](#)
- struct `netc_tb_tgs_gcl_t`
Time Gate Scheduling table entry gate control list structure. [More...](#)
- struct `netc_tb_et_cfge_t`
Egress Treatment table config element. [More...](#)
- struct `netc_tb_et_req_data_t`
Egress Treatment table request data buffer. [More...](#)
- struct `netc_tb_et_rsp_data_t`
Egress Treatment table request response data buffer. [More...](#)
- struct `netc_tb_et_data_t`
Egress Treatment table data buffer. [More...](#)
- struct `netc_tb_et_config_t`
Egress Treatment table entry config. [More...](#)
- struct `netc_tb_etmcq_cfge_t`
ETM Class Queue table config element. [More...](#)
- struct `netc_tb_etmcq_stse_t`
ETM Class Queue table statistic element. [More...](#)
- struct `netc_tb_etmcq_req_data_t`
ETM Class Queue table request data buffer. [More...](#)
- struct `netc_tb_etmcq_rsp_data_t`
ETM Class Queue table request response data buffer. [More...](#)
- struct `netc_tb_etmcq_data_t`
ETM Class Queue table data buffer. [More...](#)
- struct `netc_tb_etmcq_config_t`
ETM Class Queue table entry config. [More...](#)
- struct `netc_tb_etmcs_cfge_t`
ETM Class Scheduler table config element. [More...](#)
- struct `netc_tb_etmcs_req_data_t`
ETM Class Scheduler table request data buffer. [More...](#)
- struct `netc_tb_etmcs_rsp_data_t`
ETM Class Scheduler table request response data buffer. [More...](#)
- struct `netc_tb_etmcs_data_t`
ETM Class Scheduler table data buffer. [More...](#)
- struct `netc_tb_etmcs_config_t`
ETM Class Scheduler table entry config. [More...](#)
- struct `netc_tb_etmcs_cgcfge_t`
ETM Congestion Group table config element. [More...](#)
- struct `netc_tb_etmcs_stse_t`

- *ETM Congestion Group table statistic element.* [More...](#)
- struct `netc_tb_etmcg_req_data_t`
ETM Congestion Group table request data buffer. [More...](#)
- struct `netc_tb_etmcg_rsp_data_t`
ETM Congestion Group table request response data buffer. [More...](#)
- struct `netc_tb_etmcg_data_t`
ETM Congestion Group table data buffer. [More...](#)
- struct `netc_tb_etmcg_config_t`
ETM Congestion Group table entry config. [More...](#)
- struct `netc_tb_ec_stse_t`
Egress Count table statistic element. [More...](#)
- struct `netc_tb_ec_req_data_t`
Egress Count table request data buffer. [More...](#)
- struct `netc_tb_ec_rsp_data_t`
Egress Count table request response data buffer. [More...](#)
- struct `netc_tb_ec_data_t`
Egress Count table data buffer. [More...](#)
- struct `netc_tb_bp_cfge_t`
Buffer Pool table config element. [More...](#)
- struct `netc_tb_bp_bpse_t`
Buffer Pool table State Element Data. [More...](#)
- struct `netc_tb_bp_req_data_t`
Buffer Pool table request data buffer. [More...](#)
- struct `netc_tb_bp_rsp_data_t`
Buffer Pool table request response data buffer. [More...](#)
- struct `netc_tb_bp_data_t`
Buffer Pool table data buffer. [More...](#)
- struct `netc_tb_bp_config_t`
Buffer Pool table entry config. [More...](#)
- struct `netc_tb_sbp_cfge_t`
Shared Buffer Pool table config element. [More...](#)
- struct `netc_tb_sbp_sbpse_t`
Shared Buffer Pool table State Element Data. [More...](#)
- struct `netc_tb_sbp_req_data_t`
Shared Buffer Pool table request data buffer. [More...](#)
- struct `netc_tb_sbp_rsp_data_t`
Shared Buffer Pool table request response data buffer. [More...](#)
- struct `netc_tb_sbp_data_t`
Shared Buffer Pool table data buffer. [More...](#)
- struct `netc_tb_sbp_config_t`
Shared Buffer Pool table entry config. [More...](#)
- union `netc_tb_data_buffer_t`
Table common data buffer. [More...](#)
- struct `netc_cbdr_hw_t`
Register group for SI/Switch command bd ring. [More...](#)
- struct `netc_cbdr_handle_t`
Handle for common part of EP/Switch NTMP. [More...](#)

Macros

- #define `NETC_FD_EID_ENCODE_OPTION_0`(entryId) ((uint32_t)(entryId)&0xFFFFU)

- Frame Modification Entry ID encode options.
 - `#define NETC_ISI_VLAN_FRAME_KEY(valid, pcp, vid) (((uint16_t)((valid)&0x1) << 15U) | ((uint16_t)((pcp)&0x3) << 12U) | ((uint16_t)(vid)&0xFFFF))`
2 Bytes VLAN field which may be added to the frame Key
 - `#define NETC_TB_SGCL_MAX_ENTRY (0x100U)`
Stream Gate Control List Table maximum gate control list length.
 - `#define NETC_TB_FMD_UPDATE_CONFIG_LENGTH(x) (8U + (x))`
TFrame Modification Data table update config Data Buffer length, x is the number of update data bytes.
 - `#define NETC_TB_TGS_MAX_ENTRY (0x100U)`
Time Gate Scheduling Table maximum gate control list length (TGSTCAPR[MAX_GCL_LEN])
 - `#define NETC_TB_ETM_CQ_ENTRY_ID(portID, cqID) (((uint32_t)(portID) << 4U) | ((cqID)&0x7U))`
ETM Class Queue table entry ID macro, cqID is represents the Class Queue ID ,rang in 0 ~ 7, portID is Switch ID, rang in 0 ~ 4.
 - `#define NETC_TB_ETM(CG)_ENTRY_ID(portID, cgID) (((uint32_t)(portID) << 4U) | ((cgID)&0x7U))`
ETM Congestion Group table entry ID macro, cgID is represents the Congestion Group ID ,rang in 0 ~ 7, portID is Switch ID, rang in 0 ~ 4.
 - `#define NETC_TB_BP_THRESH(mant, exp) (((uint32_t)((mant)&0xFF) << 4U) | ((exp)&0xF))`
Buffer pool and shared buffer pool threshold macro, the threshold = MANT*2^{EXP}, uint is internal memory words (avergae of 20 bytes each)

Typedefs

- `typedef netc_tb_eteid_access_mode_t netc_tb_is_oeteid_mode_t`
Ingress Stream table Override ET_EID mode.
- `typedef netc_tc_sdu_type_t netc_tb_rp_sdu_type_t`
Rate Policer Table Protocol/Service Data Unit Type.
- `typedef netc_tc_sdu_type_t netc_tb_sgi_sdu_type_t`
Stream Gate Instance table Protocol/Service Data Unit Type.
- `typedef netc_fm_sqt_act_t netc_tb_fm_sqt_act_t`
Frame Modification table Sequence Tag Action.
- `typedef netc_tb_eteid_access_mode_t netc_tb_fdb_oeteid_mode_t`
FDB table entry defined the egress packet processing actions (will cover the actions which specified in the Egress Treatment table)
- `typedef netc_tb_fdb_cfge_t netc_tb_l2mcf_cfge_t`
L2 IPV4 Multicast Filter table config element.
- `typedef netc_tb_fdb_acte_t netc_tb_l2mcf_acte_t`
L2 IPV4 Multicast Filter table activity lement.

Enumerations

- enum `netc_tb_index_t` {

kNETC_TGSTable = 5U,
 kNETC_RPTable = 10U,
 kNETC_IPFTable = 13U,
 kNETC_FDBTable = 15U,
 kNETC_L2MCFTable = 16U,
 kNETC_VFTable = 18U,
 kNETC_ECQTable = 22U,
 kNETC_ECSTable = 23U,
 kNETC_ISITable = 30U,
 kNETC_ISTable = 31U,
 kNETC_ISFTable = 32U,
 kNETC_ETTable = 33U,
 kNETC_ISGTable = 34U,
 kNETC_ESRTable = 35U,
 kNETC_SGITable = 36U,
 kNETC_SGCLTable = 37U,
 kNETC_ISCTable = 38U,
 kNETC_ECTable = 39U,
 kNETC_FMTable = 40U,
 kNETC_BPTable = 41U,
 kNETC_SBPTable = 42U,
 kNETC_ECGTable = 43U,
 kNETC_FMDFTable = 44U }

Table index.

- enum `netc_tb_cmd_t` {

kNETC_DeleteEntry = 0x1U,
 kNETC_UpdateEntry = 0x2U,
 kNETC_QueryEntry = 0x4U,
 kNETC_QueryAndDeleteEntry = 0x5U,
 kNETC_QueryAndUpdateEntry = 0x6U,
 kNETC_AddEntry = 0x8U,
 kNETC_AddOrUpdateEntry = 0xAU,
 kNETC_AddAndQueryEntry = 0xCU,
 kNETC_AddQueryAndUpdateEntry }

Table management command operations.

- enum `netc_tb_access_mode_t` {

kNETC_EntryIDMatch = 0U,
 kNETC_ExactKeyMatch,
 kNETC_Search,
 kNETC_TernaryKeyMatch }

Table Access Method.

- enum `netc_cbd_version_t` {

kNETC_NtmpV1_0 = 0U,

```
kNETC_NtmpV2_0 }  
    NTMP version.  
• enum netc\_cmd\_error\_t {
```

kNETC_FormatError = 0x1U,
kNETC_SizeError = 0x2U,
kNETC_AccessError = 0x4U,
kNETC_ClassError = 0x8U,
kNETC_IntegrityError = 0x10U,
kNETC_InvTableID = 0x80U,
kNETC_InvAccMethod = 0x81U,
kNETC_TableIdxOutRange = 0x82U,
kNETC_DBNotEnough = 0x83U,
kNETC_InvCmd = 0x84U,
kNETC_ReqDBError = 0x85U,
kNETC_MultiBitError = 0x86U,
kNETC_HashEntryLimit = 0x87U,
kNETC_HashChainLimit = 0x88U,
kNETC_InvHWGenEntryID = 0x89U,
kNETC_SrchResDBNotEnough = 0x8AU,
kNETC_CmdIdxTableWithITM = 0x8BU,
kNETC_InvQueryAction = 0x8CU,
kNETC_InvTableAccPrivilege = 0x8DU,
kNETC_ReadSysBusErr = 0x8EU,
kNETC_WriteSysBusErr = 0x8FU,
kNETC_ClientErr = 0x90U,
kNETC_TGSCmdIssue = 0xD0U,
kNETC_TGSUpdateExistList = 0xD1U,
kNETC_TGSUpdateOverLength = 0xD2U,
kNETC_TGSUpdateOverSize = 0xD3U,
kNETC_TGSEntryNotEnough = 0xD4U,
kNETC_TGSUpdateNSList = 0xD5U,
kNETC_TGSUpdateEarlierStartTime = 0xD6U,
kNETC_TGSUpdateOverflowCycle = 0xD7U,
kNETC_TGSQueryBeforeListActive = 0xD8U,
kNETC_TGSUpdateInvGateValue = 0xD9U,
kNETC_RPSDUTypeOutRange = 0x120U,
kNETC_IPFInvHR = 0x150U,
kNETC_IPFEntryNotFit = 0x151U,
kNETC_IPFWithoutSTSE = 0x152U,
kNETC_IPFInvRPP = 0x154U,
kNETC_IPFFLTATGTOutRange = 0x155U,
kNETC_IPFInvSwtFLTA = 0x156U,
kNETC_IPFInvEnetcFLTA = 0x157U,
kNETC_FDBReachPortLimit = 0x171U,
kNETC_FDBReachSwtLimit = 0x172U,
kNETC_FDBInvEPORT = 0x173U,
kNETC_FDBETEIDOutRange = 0x174U,
kNETC_FDBParityErr = 0x175U,
kNETC_L2MCFInvEPORT = 0x181U,
kNETC_L2MCFETEIDOutRange = 0x182U,
kNETC_L2MCFInvKEYTYPE = 0x183U,
kNETC_VFBASEETEIDOutRange = 0x1A0U,

- ```

kNETC_BPSBPEIDOutRange = 0x310U }

Table command response error status.
• enum netc_fm_vlan_ud_act_t {
 kNETC_NoUDVlanAction = 0U,
 kNETC_ReplVlanPcpAndDei,
 kNETC_DelVlan }
 Frame Modification VLAN Update/Delete Action.
• enum netc_fm_sqt_act_t {
 kNETC_NoSqtAction = 0U,
 kNETC_ReplaceRTag }
 Frame Modification Sequence Tag Action.
• enum netc_fm_vlan_ar_act_t {
 kNETC_AddCVlanPcpAndDei = 0U,
 kNETC_AddSVlanPcpAndDei,
 kNETC_ReplVidOnly,
 kNETC_ReplVidPcpAndDei }
 Frame Modification VLAN Add/Replace Action.
• enum netc_tb_eteid_access_mode_t {
 kNETC_NoETAccess = 0x0U,
 kNETC_SinglePortETAccess,
 kNETC_MulitPortPackedETAccess,
 kNETC_MulitPortAbsETAccess }

Define FDB/L2MCF/IS table entry access the primary Egress Treatment table entry group mode.
• enum netc_tb_ipf_update_action_t {
 kNETC_IPFCfgEUpdate = 0x1U,
 kNETC_IPFStsEUpdate = 0x2U }

Ingress Port Filter Table Update Actions.
• enum netc_tb_ipf_attr_mask_t {
 kNETC_IPFSwtPortMasMask = 0x1U,
 kNETC_IPFEthernetMask = 0x2U,
 kNETC_IPFOuterVlanMask = 0x4U,
 kNETC_IPFIInnerVlanMask = 0x8U,
 kNETC_IPFSeqTagMask = 0x70U,
 kNETC_IPFIpHeaderMask = 0x80U,
 kNETC_IPFIpVersionMask = 0x100U,
 kNETC_IPFIpExtMask = 0x200U,
 kNETC_IPFL4HeaderMask = 0x400U,
 kNETC_IPFWakeOnLanMask = 0x800U }

Ingress Port Filter frame attribute mask.
• enum netc_tb_ipf_seq_tag_t {
 kNETC_IPFNoRtag = 0x0U,
 kNETC_IPFDraftRtag,
 kNETC_IPFRtag,
 kNETC_IPFHsrTag }

Ingress Port Filter frame attribute Sequence Tag Code.
• enum netc_tb_ipf_14_header_t {

```

```
kNETC_IPFOtherL4,
kNETC_IPFTcp,
kNETC_IPFUpd,
kNETC_IPFSctp }
```

*Ingress Port Filter frame attribute L4 Header Code.*

- enum `netc_tb_ipf_forward_action_t` {
 kNETC\_IPFFForwardDiscard = 0U,
 kNETC\_IPFFForwardPermit,
 kNETC\_IPFRedirectToMgmtPort,
 kNETC\_IPFCopyToMgmtPort }

*Ingress port filter forwarding Action.*

- enum `netc_tb_ipf_filter_action_t` {
 kNETC\_IPFNoAction = 0U,
 kNETC\_IPFWithRatePolicer,
 kNETC\_IPFWithIngressStream,
 kNETC\_IPFWithL2Filtering }

*Ingress port filter Filter Action.*

- enum `netc_tb_isi_key_type` {
 kNETC\_KCRule0 = 0U,
 kNETC\_KCRule1,
 kNETC\_KCRule2,
 kNETC\_KCRule3 }

*Stream identification table key type.*

- enum `netc_tb_is_isq_action_t` {
 kNETC\_ISNotPerformFRER = 0,
 kNETC\_ISPerformFRER }

*Ingress Stream table Ingress Sequence Action.*

- enum `netc_tb_is_forward_action_t` {
 kNETC\_ISDiscard = 0U,
 kNETC\_ISRedirectToMgmtPort = 1U,
 kNETC\_ISAllow = 1U,
 kNETC\_ISAllowWithSIMap = 2U,
 kNETC\_ISStreamForward = 2U,
 kNETC\_ISBridgeForward,
 kNETC\_ISCopyToMgmtPortAndStream,
 kNETC\_ISCopyToMgmtPortAndBridge }

*Ingress Stream table forwarding Action.*

- enum `netc_tb_is_ctd_mode_t` {
 kNETC\_ISNoCTD = 0x0U,
 kNETC\_ISSinglePortCTD,
 kNETC\_ISAllPortCTD }

*Ingress Stream table Cut-Through Disable mode.*

- enum `netc_tb_rp_update_action_t` {
 kNETC\_RPCfgEUpdate = 0x1U,
 kNETC\_RPFcEUpdate = 0x2U,
 kNETC\_RPPsEUpdate = 0x4U,

- ```

kNETC_RPStsEUpdate = 0x8U }

Rate Policer Table Update Actions.
• enum netc_tb_sgi_update_action_t {
    kNETC_SGIAcEUpdate = 0x1U,
    kNETC_SGICfgEUpdate = 0x2U,
    kNETC_SGISgisEUpdate = 0x4U }

Stream Gate Instance table Update Actions.
• enum netc_tb_sgi_state_t {
    kNETC_GSNotOper = 0x0U,
    kNETC_GSUseDefaultParam,
    kNETC_GSUseDefUntilAdminAct,
    kNETC_GSUseOperUntilAdminAct,
    kNETC_GSUseOperList }

Stream Gate Instance State.
• enum netc_tb_fm_layer2_act_t {
    kNETC_UseL2HeaderAct = 0U,
    kNETC_UseSpecPayload }

Frame Modification table Layer 2 Actions.
• enum netc_tb_fm_mac_header_act_t {
    kNETC_NoMacAction = 0U,
    kNETC_ReplSmac = 2U,
    kNETC_ReplSmacAndDmacAct1,
    kNETC_ReplSmacAndDmacAct2,
    kNETC_ReplDmac,
    kNETC_SwapDmacAndSmac }

Frame Modification table Layer 2 Header MAC Actions.
• enum netc_tb_fm_vlan_header_act_t {
    kNETC_NoVlanAction = 0U,
    kNETC_DelOuterVlan,
    kNETC_AddOuterVlan,
    kNETC_ReplOuterVlan }

Frame Modification table Layer 2 VLAN Actions.
• enum netc_tb_fm_outer_vid_act_t {
    kNETC_UseFrameVid = 0U,
    kNETC_UseSpecVid }

Frame Modification table Layer 2 outer VLAN VID Actions.
• enum netc_tb_fm_outer_tpid_act_t {
    kNETC_UseFrameTpid = 0U,
    kNETC_UseStdCVlan,
    kNETC_UseStdSVlan,
    kNETC_UseCustomCVlan,
    kNETC_UseCustomSVlan }

Frame Modification table Outer TPID action.
• enum netc_tb_fm_outer_pcp_act_t {
    kNETC_UseFramePcp = 0U,
    kNETC_UseSpecPcp,
    kNETC_UseFramePcpMap,

```

- ```

kNETC_UseQosMap }

Frame Modification table Outer PCP action.
• enum netc_tb_fm_outer_dei_act_t {
 kNETC_UseFrameDei = 0U,
 kNETC_UseSpecDei,
 kNETC_UseDrMap }

Frame Modification table Outer DEI action.
• enum netc_tb_fm_payload_act_t {
 kNETC_NoAction = 0U,
 kNETC_ReplAllEthPld,
 kNETC_ReplPldWithOffset }

Frame Modification table Payload Actions.
• enum netc_tb_fdb_update_action_t {
 kNETC_FDBCfgEUpdate = 0x1U,
 kNETC_FDBActEUpdate = 0x2U }

FDB table Update Actions.
• enum netc_tb_fdb_ctd_mode_t {
 kNETC_FDBNoCTD = 0x0U,
 kNETC_FDBSinglePortCTD = 0x1U,
 kNETC_FDBAllPortCTD = 0x2U }

FDB table Cut-Through Disable mode.
• enum netc_tb_fdb_sc_keye_mc_t {
 kNETC_FDBKeyeMacthAny = 0x0,
 kNETC_FDBKeyeMacthFID = 0x1,
 kNETC_FDBKeyeMacthMacMulticast = 0x2,
 kNETC_FDBKeyeMacthBoth = 0x3 }

FDB table search criteria Key Element Match Criteria.
• enum netc_tb_fdb_sc_cfge_mc_t {
 kNETC_FDBCfgeMacthAny = 0x0,
 kNETC_FDBCfgeMacthDynamic = 0x1,
 kNETC_FDBCfgeMacthPortBitmap = 0x2,
 kNETC_FDBCfgeMacthBoth = 0x3 }

FDB table search criteria Configuration Element Match Criteria.
• enum netc_tb_fdb_sc_akte_mc_t {
 kNETC_FDBActeMacthAny = 0x0,
 kNETC_FDBActeMatchExact = 0x1 }

FDB table search criteria Activity Element Match Criteria.
• enum netc_tb_l2mcf_key_type_t {
 kNETC_IPv4ASMKey = 0U,
 kNETC_IPv4SSMKey }

L2 IPV4 Multicast Filter table key type.
• enum etc_tb_l2mcf_sc_keye_mc_t {
 kNETC_L2MCFKeyeMacthAny = 0x0,
 kNETC_L2MCFKeyeMacthFID = 0x1 }

L2 IPV4 Multicast Filter table search criteria Key Element Match Criteria.
• enum etc_tb_l2mcf_sc_cfge_mc_t {

```

```
kNETC_L2MCFCfgeMacthAny = 0x0,
kNETC_L2MCFCfgeMacthDynamic = 0x1,
kNETC_L2MCFCfgeMacthPortBitmap = 0x2,
kNETC_L2MCFCfgeMacthBoth = 0x3 }
```

*L2 IPv4 Multicast Filter table search criteria Configuration Element Match Criteria.*

- enum `etc_tb_l2mcf_sc_acte_mc_t` {
 kNETC\_L2MCFActeMacthAny = 0x0,
 kNETC\_L2MCFActeMatchExact = 0x1 }

*FDB table search criteria Activity Element Match Criteria.*

- enum `netc_tb_iseqg_sqtag_t` {
 kNETC\_SqDraftRTag = 1U,
 kNETC\_SqRTag,
 kNETC\_SqHsrTag }

*Sequence Tag Type.*

- enum `netc_tb_eseqr_sqtag_t` {
 kNETC\_AcceptAnyTag = 0U,
 kNETC\_AcceptDraftRTag = 1U,
 kNETC\_AcceptRTag,
 kNETC\_AcceptHsrTag }

*Egress Sequence Recovery table Sequence Tag Type.*

- enum `netc_tb_tgs_entry_id_t` {
 kNETC\_TGSSwtPort0 = 0U,
 kNETC\_TGSSwtPort1,
 kNETC\_TGSSwtPort2,
 kNETC\_TGSSwtPort3,
 kNETC\_TGSSwtPort4,
 kNETC\_TGSEnetc0Port,
 kNETC\_TGSEnetc1Port }

*Time Gate Scheduling table entry ID for switch and ENETC.*

- enum `netc_tb_tgs_gate_type_t` {
 kNETC\_SetGateStates = 0U,
 kNETC\_SetAndHoldMac,
 kNETC\_SetAndReleaseMac }

*Administrative gate operation type.*

- enum `netc_tb_et_efm_mode_t` {
 kNETC\_NormalMode = 0U,
 kNETC\_L2Act1,
 kNETC\_PldAct1 }

*Egress Frame Modification entry mode.*

- enum `netc_tb_et_esq_act_t` {
 kNETC\_NoEsqAction = 0U,
 kNETC\_HasEsqAction = 2U }

*Egress Sequence Actions.*

- enum `netc_tb_et_ec_act_t` {
 kNETC\_NoEcCounter = 0U,
 kNETC\_HasEcCounter }

*Egress Counter Action.*

- enum `netc_tb_etmcq_update_action_t` {
   
  `kNETC_CQCfgEUpdate` = 0x1U,
   
  `kNETC_CQStsEUpdate` }
   
    *ETM Class Queue table Update Actions.*
- enum `netc_tb_etmcs_entry_id_t` {
   
  `kNETC_CSSwtPort0` = 0U,
   
  `kNETC_CSSwtPort1`,
   
  `kNETC_CSSwtPort2`,
   
  `kNETC_CSSwtPort3`,
   
  `kNETC_CSSwtPort4` }
   
    *ETM Class Scheduler table entry ID.*
- enum `netc_tb_etmcs_ca_assg_t` {
   
  `kNETC_CQ7AssignToSchedIn15` = 0U,
   
  `kNETC_CQ7AssignToSchedIn14`,
   
  `kNETC_CQ7AssignToSchedIn13`,
   
  `kNETC_CQ7AssignToSchedIn12`,
   
  `kNETC_CQ7AssignToSchedIn11`,
   
  `kNETC_CQ7AssignToSchedIn10`,
   
  `kNETC_CQ7AssignToSchedIn9`,
   
  `kNETC_CQ7AssignToSchedIn8`,
   
  `kNETC_CQ7AssignToSchedIn7` }
   
    *ETM Class Scheduler table Class queue assignment to scheduler inputs mode.*
- enum `netc_tb_bp_fc_cfg_t` {
   
  `kNETC_FlowCtrlDisable` = 0U,
   
  `kNETC_FlowCtrlWithBP`,
   
  `kNETC_FlowCtrlWithSBP`,
   
  `kNETC_FlowCtrlWithBPAAndSBP` }
   
    *Buffer Pool Flow Control (FC) Configuration.*

## Functions

- `status_t NETC_CmdBDRInit (netc_cbdr_hw_t *base, const netc_cmd_bdr_config_t *config)`
  
    *Initialize the command BD ring.*
- `status_t NETC_CmdBDRDeinit (netc_cbdr_hw_t *base)`
  
    *Deinitialize the command BD ring.*
- `status_t NETC_CmdBDSendCommand (netc_cbdr_hw_t *base, netc_cmd_bdr_t *cbdr, netc_cmd_bd_t *cbd, netc_cbd_version_t version)`
  
    *Send the Command Buffer Descriptor to operate on a NTMP table.*
- `status_t NETC_AddIPFTableEntry (netc_cbdr_handle_t *handle, netc_tb_ipf_config_t *config, uint32_t *entryID)`
  
    *Add entry into the ingress Port Filter Table.*
- `status_t NETC_UpdateIPFTableEntry (netc_cbdr_handle_t *handle, uint32_t entryID, netc_tb_ipf_cfge_t *cfg)`
  
    *Update entry in the ingress Port Filter Table.*
- `status_t NETC_QueryIPFTableEntry (netc_cbdr_handle_t *handle, uint32_t entryID, netc_tb_ipf_config_t *config)`
  
    *Query entry in the ingress Port Filter Table.*

- **status\_t NETC\_DelIPFTableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Delete an entry in the ingress Port Filter Table.*
- **status\_t NETC\_ResetIPFMatchCounter (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Reset the counter of an ingress port filter Table entry.*
- **status\_t NETC\_GetIPFMatchedCount (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID, uint64\_t \*count)**  
*Get the matched count of an ingress port filter Table entry.*
- **status\_t NETC\_AddISITableEntry (netc\_cbdr\_handle\_t \*handle, netc\_tb\_isi\_config\_t \*config, uint32\_t \*entryID)**  
*Add entry into Ingress Stream Identification table.*
- **status\_t NETC\_DelISITableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Delete an entry in Ingress stream identification table.*
- **status\_t NETC\_QueryISITableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID, netc\_tb\_isi\_config\_t \*config)**  
*Query Ingress Stream Identification table.*
- **status\_t NETC\_QueryISITableEntryWithKey (netc\_cbdr\_handle\_t \*handle, netc\_tb\_isi\_keye\_t \*keye, netc\_tb\_isi\_rsp\_data\_t \*rsp)**  
*Query Ingress Stream Identification table with key.*
- **status\_t NETC\_AddOrUpdateISTableEntry (netc\_cbdr\_handle\_t \*handle, netc\_tb\_is\_config\_t \*config, bool isAdd)**  
*Add or update entry in Ingress Stream table.*
- **status\_t NETC\_QueryISTableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID, netc\_tb\_is\_config\_t \*config)**  
*Query Ingress Stream table.*
- **status\_t NETC\_DelISTableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Delete an entry in Ingress stream table.*
- **status\_t NETC\_AddISFTableEntry (netc\_cbdr\_handle\_t \*handle, netc\_tb\_isf\_config\_t \*config, uint32\_t \*entryID)**  
*Add entry into ingress stream filter table.*
- **status\_t NETC\_UpdateISFTableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID, netc\_tb\_isf\_cfg\_e\_t \*cfg)**  
*Update entry into ingress stream filter table.*
- **status\_t NETC\_DelISFTableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Delete an entry in Ingress stream filter table.*
- **status\_t NETC\_QueryISFTableEntry (netc\_cbdr\_handle\_t \*handle, netc\_tb\_isf\_keye\_t \*keye, netc\_tb\_isf\_rsp\_data\_t \*rsp)**  
*Query entry from the Ingress stream filter table.*
- **status\_t NETC\_AddISCTableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Add entry in ingress stream count table.*
- **status\_t NETC\_GetISCStatistic (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID, netc\_tb\_isc\_stse\_t \*statistic)**  
*Get ingress stream count statistic.*
- **status\_t NETC\_ResetISCStatistic (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Reset the count of the ingress stream count.*
- **status\_t NETC\_AddOrUpdateSGITableEntry (netc\_cbdr\_handle\_t \*handle, netc\_tb\_sgi\_config\_t \*config, bool isAdd)**  
*Add or update entry in stream gate instance table.*
- **status\_t NETC\_DelSGITableEntry (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID)**  
*Delete entry in the stream gate instance table.*
- **status\_t NETC\_GetSGIState (netc\_cbdr\_handle\_t \*handle, uint32\_t entryID, netc\_tb\_sgi\_sgise\_t**

`*status)`

*Get statistic of specified stream gate instance table entry.*

- `status_t NETC_QuerySGITableEntry (netc_cbdr_handle_t *handle, uint32_t entryID, netc_tb_sg_i_rsp_data_t *rsp)`

*Query entry from the stream gate instance table.*

- `status_t NETC_AddSGCLTableEntry (netc_cbdr_handle_t *handle, netc_tb_sgcl_gcl_t *config)`

*Add entry into Stream Gate Control List Table.*

- `status_t NETC_DelSGCLTableEntry (netc_cbdr_handle_t *handle, uint32_t entryID)`

*Delete entry of Stream Gate Control List Table.*

- `status_t NETC_GetSGCLGateList (netc_cbdr_handle_t *handle, netc_tb_sgcl_gcl_t *gcl, uint32_t length)`

*Get Stream Gate Control List Table entry gate control list.*

- `status_t NETC_GetSGCLState (netc_cbdr_handle_t *handle, uint32_t entryID, netc_tb_sgcl_sgclse_t *state)`

*Get state (ref count) for Stream Gate Control List table entry.*

- `status_t NETC_QueryRPTableEntry (netc_cbdr_handle_t *handle, uint32_t entryID, netc_tb_rp_rsp_data_t *rsp)`

*Query entry from the Rate Policer table.*

- `status_t NETC_AddOrUpdateRPTableEntry (netc_cbdr_handle_t *handle, netc_tb_rp_config_t *config, netc_tb_cmd_t cmd)`

*Add or update entry in Rate Policer table.*

- `status_t NETC_DelRPTableEntry (netc_cbdr_handle_t *handle, uint32_t entryID)`

*Delete entry in the Rate Policer table.*

- `status_t NETC_GetRPStatistic (netc_cbdr_handle_t *handle, uint32_t entryID, netc_tb_rp_stse_t *stat)`

*Get statistic of specified Rate Policer table entry.*

- `status_t NETC_ConfigTGSAdminList (netc_cbdr_handle_t *handle, netc_tb_tgs_gcl_t *config)`

*Config the QBV (Time Gate Scheduling)*

- `status_t NETC_GetTGSOperationList (netc_cbdr_handle_t *handle, netc_tb_tgs_gcl_t *gcl, uint32_t length)`

*Get time gate table operation list.*

## 29.5.11.2 Data Structure Documentation

### 29.5.11.2.1 union netc\_cmd\_bd\_t

#### Field Documentation

##### (1) uint64\_t netc\_cmd\_bd\_t::addr

Data.

- (2) `netc_tb_cmd_t netc_cmd_bd_t::cmd`
- (3) `netc_tb_access_mode_t netc_cmd_bd_t::accessType`
- (4) `uint32_t netc_cmd_bd_t::__pad1__`
- (5) `uint32_t netc_cmd_bd_t::version`
- (6) `uint32_t netc_cmd_bd_t::enComplInt`
- (7) `uint32_t netc_cmd_bd_t::resReady`
- (8) `uint32_t netc_cmd_bd_t::npf`
- (9) `uint32_t netc_cmd_bd_t::numMatched`
- (10) `uint32_t netc_cmd_bd_t::error`
- (11) `uint32_t netc_cmd_bd_t::en`
- (12) `uint32_t netc_cmd_bd_t::siBitMap`
- (13) `uint32_t netc_cmd_bd_t::index`
- (14) `uint32_t netc_cmd_bd_t::cmd`
- (15) `uint32_t netc_cmd_bd_t::__pad2__`
- (16) `uint32_t netc_cmd_bd_t::status`
- (17) `uint32_t netc_cmd_bd_t::ci`
- (18) `uint32_t netc_cmd_bd_t::sf`

### 29.5.11.2.2 struct netc\_cmd\_bdr\_config\_t

#### Data Fields

- `netc_cmd_bd_t * bdBase`  
*BDR base address which shall be 128 bytes aligned.*
- `uint16_t bdLength`  
*Size of BD ring which shall be multiple of 8 BD.*
- `bool enCompInt`  
*Enable/Disable command BD completion interrupt.*

### 29.5.11.2.3 struct netc\_cmd\_bdr\_t

#### Data Fields

- `netc_cmd_bd_t * bdBase`

- **uint16\_t bdLength**  
*Size of BD ring.*
- **uint16\_t producerIndex**  
*Current index for execution.*
- **uint16\_t cleanIndex**  
*Current index for cleaning.*
- **bool bdrEnable**  
*Current command BD ring is enable or not.*

## Field Documentation

- (1) **uint16\_t netc\_cmd\_bdr\_t::producerIndex**
- (2) **uint16\_t netc\_cmd\_bdr\_t::cleanIndex**
- (3) **bool netc\_cmd\_bdr\_t::bdrEnable**

### 29.5.11.2.4 struct netc\_tb\_common\_header\_t

#### Data Fields

- **uint32\_t updateActions:** 16  
*Update Actions.*
- **uint32\_t queryActions:** 4  
*Query Actions.*

### 29.5.11.2.5 struct netc\_tb\_ipf\_keye\_t

#### Data Fields

- **uint16\_t precedence**  
*Precedence value of an entry.*
- **struct {**
  - uint16\_t swtPortMas:** 1  
*Switch port masquerading, applicable only if the incoming port is designated as a switch management port.*
  - uint16\_t ethernet:** 1  
*Ethernet type Present.*
  - uint16\_t outerVlan:** 1  
*Outer VLAN Present.*
  - uint16\_t innerVlan:** 1  
*Inner VLAN Present.*
  - netc\_tb\_ipf\_seq\_tag\_t seqTag:** 3  
*Sequence Tag Code.*
  - uint16\_t ipHeader:** 1  
*IP Header Present.*
  - uint16\_t ipVersion:** 1  
*0b = IPv4, 1b = IPv6*

```

 uint16_t ipExt: 1
 IPv4 option / IPv6 extension present.
 netc_tb_ipf_l4_header_t l4Header: 2
 L4 Header code.
 uint16_t wakeOnLan: 1
 Wake-on-LAN Magic Packet Present.
} frameAttr

```

- *Frame Attribute flags.*
- uint16\_t **frameAttrMask**  
*Frame attribute mask, set with OR of [netc\\_tb\\_ipf\\_attr\\_mask\\_t](#).*
- uint16\_t **dscp**: 6  
*Differentiated Services Code Point.*
- uint16\_t **dscpMask**: 6  
*Differentiated Services Code Point Mask.*
- uint16\_t **srcPort**: 5  
*Source Port ID.*
- uint16\_t **srcPortMask**: 5  
*Source Port ID Mask.*
- uint16\_t **outerVlanTCI**  
*Outer VLAN Tag Control Information.*
- uint16\_t **outerVlanTCIMask**  
*Outer VLAN Tag Control Information Mask.*
- uint8\_t **dmac** [6]  
*Destination MAC Address.*
- uint8\_t **dmacMask** [6]  
*Destination MAC Address Mask.*
- uint8\_t **smac** [6]  
*Source MAC Address.*
- uint8\_t **smacMask** [6]  
*Source MAC Address Mask.*
- uint16\_t **innerVlanTCI**  
*Inner VLAN Tag Control Information.*
- uint16\_t **innerVlanTCIMask**  
*Inner VLAN Tag Control Information Mask.*
- uint16\_t **etherType**  
*2-byte EtherType*
- uint16\_t **etherTypeMask**  
*EtherType Mask.*
- uint8\_t **IPProtocol**  
*IP Protocol.*
- uint8\_t **IPProtocolMask**  
*IP Protocol Mask.*
- uint8\_t **srcIPAddr** [16]  
*IP Source Address, Bits 127-0: IPv6 source address, Bits 127-96: IPv4 source address.*
- uint8\_t **srcIPAddrMask** [16]  
*IP Source Address Mask.*
- uint16\_t **l4SrcPort**  
*L4 Source Port.*
- uint16\_t **l4SrcPortMask**

- **L4 Source Port Mask.**
- **uint8\_t destIPAddr [16]**  
*IP Destination Address, Bits 127-0: IPv6 source address, Bits 127-96: IPv4 source address.*
- **uint8\_t destIPAddrMask [16]**  
*IP Destination Address Mask.*
- **uint16\_t l4DestPort**  
*L4 Destination Port.*
- **uint16\_t l4DestPortMask**  
*L4 Destination Port Mask.*
- **uint8\_t data**  
*Payload Byte n.*
- **uint8\_t mask**  
*Payload Byte n Mask.*

### 29.5.11.2.6 struct netc\_tb\_ipf\_cfge\_t

#### Data Fields

- **uint32\_t ipv: 4**  
*Internal Priority Value.*
- **uint32\_t oipv: 1**  
*Overwrite IPV.*
- **uint32\_t dr: 2**  
*Drop Resilience.*
- **uint32\_t odr: 1**  
*Overwrite DR.*
- **netc\_tb\_ipf\_forward\_action\_t flfa: 2**  
*Filter Forwarding action.*
- **uint32\_t imire: 1**  
*Ingress Mirroring Enable.*
- **uint32\_t wolte: 1**  
*Wake-onLAN trigger enable.*
- **netc\_tb\_ipf\_filter\_action\_t flta: 2**  
*Filter Action.*
- **uint32\_t rpr: 2**  
*Relative Precedent Resolution.*
- **uint32\_t ctd: 1**  
*Cut through disable.*
- **netc\_host\_reason\_t hr: 4**  
*Host Reason metadata when frame is redirected/copied to the switch management port.*
- **uint32\_t timecape: 1**  
*Timestam capture enable.*
- **uint32\_t fltaTgt**  
*Target for selected switch forwarding action or filter action.*

**Field Documentation**(1) `netc_tb_ipf_forward_action_t netc_tb_ipf_cfg_t::fltfa`(2) `netc_tb_ipf_filter_action_t netc_tb_ipf_cfg_t::flta`(3) `uint32_t netc_tb_ipf_cfg_t::ctd`**29.5.11.2.7 struct netc\_tb\_ipf\_stse\_t****Data Fields**

- `uint32_t matchCount [2]`

*A count of how many times this entry has been matched.*

**Field Documentation**(1) `uint32_t netc_tb_ipf_stse_t::matchCount[2]`**29.5.11.2.8 struct netc\_tb\_ipf\_req\_data\_t****29.5.11.2.9 struct netc\_tb\_ipf\_rsp\_data\_t****Data Fields**

- `uint32_t entryID`  
*Present only for commands which perform a query.*
- `netc_tb_ipf_keye_t keye`  
*Present only for commands which perform a query.*
- `netc_tb_ipf_stse_t stse`  
*Present only for commands which perform a query.*
- `netc_tb_ipf_cfg_t cfg`  
*Present only for commands which perform a query.*

**29.5.11.2.10 struct netc\_tb\_ipf\_data\_t****29.5.11.2.11 struct netc\_tb\_ipf\_config\_t****29.5.11.2.12 struct netc\_tb\_isi\_keye\_t****Data Fields**

- `netc_tb_isi_key_type keyType`: 2  
*Define the key type used for the current isi entry.*
- `uint8_t srcPortID`: 5  
*Source Port ID, used when kc portp filed is 1.*
- `uint8_t spm`: 1  
*Source Port Masquerading, used when kc spm filed is 1.*
- `uint8_t framekey [16]`  
*Frame portion of the key.*

**Field Documentation**

(1) `uint8_t netc_tb_isi_keye_t::srcPortID`

Only for SWITCH

(2) `uint8_t netc_tb_isi_keye_t::spm`

Only for SWITCH

(3) `uint8_t netc_tb_isi_keye_t::framekey[16]`

**29.5.11.2.13 struct netc\_tb\_isi\_cfge\_t****Data Fields**

- `uint32_t iSEID`

*Ingress stream entry ID, 0xFFFFFFFF means NULL.*

**29.5.11.2.14 struct netc\_tb\_isi\_req\_data\_t****29.5.11.2.15 struct netc\_tb\_isi\_rsp\_data\_t****Data Fields**

- `uint32_t entryID`

*Only present for query command.*

- `netc_tb_isi_keye_t keye`

*Only present for query command.*

- `netc_tb_isi_cfge_t cfge`

*Only present for query command.*

**29.5.11.2.16 struct netc\_tb\_isi\_data\_t****29.5.11.2.17 struct netc\_tb\_isi\_config\_t****29.5.11.2.18 struct netc\_tb\_isi\_cfge\_t****Data Fields**

- `uint32_t sfe: 1`

*Stream Filtering Enable.*

- `uint32_t ipv: 4`

*Internal Priority Value, active when opiv is set to 1.*

- `uint32_t oipv: 1`

*Override internal priority value.*

- `uint32_t dr: 2`

*Drop Resilience, active when odr is set to 1.*

- `uint32_t odr: 1`

- **Overwrite DR.**
- **uint32\_t imire:** 1  
*Ingress Mirroring Enable, not applicable to ENETC.*
- **uint32\_t timecape:** 1  
*Timestamp Capture Enable, not applicable to ENETC.*
- **uint32\_t sppd:** 1  
*Source Port Pruning Disable, not applicable to ENETC.*
- **netc\_tb\_is\_isq\_action\_t isqa:** 2  
*Ingress Sequence Action, not applicable to ENETC.*
- **uint32\_t orp:** 1  
*Override Rate Policer ID.*
- **uint32\_t osgi:** 1  
*Override stream gate instance entry id (default is NULL)*
- **netc\_host\_reason\_t hr:** 4  
*Host Reason when frame is redirected (fa = 01b) to the switch management port or copied to the switch management port (fa = 100b or 101b), value specified has to be a software defined Host Reason (8-15).*
- **netc\_tb\_is\_forward\_action\_t fa:** 3  
*Forwad Option.*
- **netc\_tc\_sdu\_type\_t sduType:** 2  
*Service Data Unit Type to user for MSDU.*
- **uint32\_t msdu:** 16  
*Maximum Service Data Unit.*
- **uint32\_t ifmeLenChange:** 7  
*Ingress Frame Modification Entry Frame Length Change, specified in unit of bytes using a 2's complement notation.*
- **uint32\_t eport:** 5  
*Egress Port which need do egress packet processing, active when oeteid is set to 1, not applicable to ENETC.*
- **netc\_tb\_is\_oeteid\_mode\_t oETEID:** 2  
*Override ET\_EID (Egress Treatment table entry, which specified egress packet processing actions)*
- **netc\_tb\_is\_ctd\_mode\_t ctd:** 2  
*Cut-Through Disable mode, valid if fa = 010b ~ 101b.*
- **uint32\_t isqEID**  
*Ingress Sequence Generation Entry ID, Valid when isqa is set to 1.*
- **uint32\_t rpEID**  
*Rate Policer Entry ID, Valid when orp =1.*
- **uint32\_t sgiEID**  
*Stream Gate Instance Entry ID, Valid when osgi =1.*
- **uint32\_t ifmEID**  
*Ingress Frame Modification Entry ID.*
- **uint32\_t etEID**  
*Base Egress Treatment Entry ID for primary Egress Treatment group, Valid alid if fa = 010b ~ 101b.*
- **uint32\_t iscEID**  
*Ingress Stream counter Index.*
- **uint32\_t ePortBitmap**  
*Egress Port bitmap, identifies the ports to which the frame is to be forwarding or ET applicability port bitmap when oETEID = 10b.*
- **uint32\_t siMap:** 16  
*Station Interface Map, only valid for ENETC function when fa field is set to 10b.*

**Field Documentation**(1) **netc\_host\_reason\_t netc\_tb\_is\_cfge\_t::hr**(2) **uint32\_t netc\_tb\_is\_cfge\_t::isqEID**

0xFFFF\_FFFF is NULL. Not applicable to ENETC

(3) **uint32\_t netc\_tb\_is\_cfge\_t::rpEID**

0xFFFF\_FFFF is NULL

(4) **uint32\_t netc\_tb\_is\_cfge\_t::sgiEID**

0xFFFF\_FFFF is NULL

(5) **uint32\_t netc\_tb\_is\_cfge\_t::ifmEID**

0xFFFF\_FFFF is NULL

(6) **uint32\_t netc\_tb\_is\_cfge\_t::etEID**

0xFFFF\_FFFF is NULL. Not applicable to ENETC

(7) **uint32\_t netc\_tb\_is\_cfge\_t::iscEID**

0xFFFF\_FFFF is NULL.

(8) **uint32\_t netc\_tb\_is\_cfge\_t::ePortBitmap**

Not applicable to ENETC

**29.5.11.2.19 struct netc\_tb\_is\_req\_data\_t****Data Fields**

- **netc\_tb\_is\_cfge\_t cfge**  
*Only perform for update or add command.*

**29.5.11.2.20 struct netc\_tb\_is\_rsp\_data\_t****Data Fields**

- **uint32\_t entryID**  
*Only perform for query command.*
- **netc\_tb\_is\_cfge\_t cfge**  
*Only perform for query command.*

**29.5.11.2.21 struct netc\_tb\_is\_data\_t****29.5.11.2.22 struct netc\_tb\_is\_config\_t****29.5.11.2.23 struct netc\_tb\_isf\_keye\_t****Data Fields**

- **uint32\_t isEID**  
*Ingress Stream Entry ID.*
- **uint8\_t pcp:** 3  
*Priority Code Point, Outer VLAN TAG PCP of the received frame.*

**29.5.11.2.24 struct netc\_tb\_isf\_cfge\_t****Data Fields**

- **uint32\_t ipv:** 4  
*Internal Priority Value, active when opiv is set to 1.*
- **uint32\_t oipv:** 1  
*Override internal priority value.*
- **uint32\_t dr:** 2  
*Drop Resilience, active when odr is set to 1.*
- **uint32\_t odr:** 1  
*Overwrite DR.*
- **uint32\_t imire:** 1  
*Ingress Mirroring Enable, not applicable to ENETC.*
- **uint32\_t timecape:** 1  
*Timestamp Capture Enable, not applicable to ENETC.*
- **uint32\_t osgi:** 1  
*Override stream gate instance entry id.*
- **uint32\_t ctd:** 1  
*Cut-Through Disable, will disable cut-through for all destined ports when set 1, not applicable to ENETC.*
- **uint32\_t orp:** 1  
*Override Rate Policer (instance) ID.*
- **netc\_tc\_sdu\_type\_t sduType:** 2  
*Service Data Unit Type to user for MSDU.*
- **uint32\_t msdu:** 16  
*Maximum Service Data Unit.*
- **uint32\_t rpEID**  
*Rate Policer Entry ID, Valid when orp =1.*
- **uint32\_t sgiEID**  
*Stream Gate Instance Entry ID, Valid when osgi =1.*
- **uint32\_t iscEID**  
*Ingress Stream counter Index.*

**Field Documentation**

(1) `uint32_t netc_tb_isf_cfge_t::rpEID`

0xFFFF\_FFFF is NULL

(2) `uint32_t netc_tb_isf_cfge_t::sgiEID`

0xFFFF\_FFFF is NULL

(3) `uint32_t netc_tb_isf_cfge_t::iscEID`

0xFFFF\_FFFF is NULL.

**29.5.11.2.25 struct netc\_tb\_isf\_req\_data\_t****Data Fields**

- `netc_tb_isf_cfge_t cfge`  
*Only perform for update or add command.*

**29.5.11.2.26 struct netc\_tb\_isf\_rsp\_data\_t****Data Fields**

- `uint32_t entryID`  
*Only perform for query command.*
- `netc_tb_isf_keye_t keye`  
*Only perform for query command.*
- `netc_tb_isf_cfge_t cfge`  
*Only perform for query command.*

**29.5.11.2.27 struct netc\_tb\_isf\_data\_t****29.5.11.2.28 struct netc\_tb\_isf\_config\_t****29.5.11.2.29 struct netc\_tb\_rp\_cfge\_t****Data Fields**

- `uint32_t cir`  
*Committed information Rate.*
- `uint32_t cbs`  
*Committed Burst Size.*
- `uint32_t eir`  
*Excess information Rate.*
- `uint32_t ebs`  
*Excess Burst Size.*

- `uint32_t mren`: 1  
*Mark All Frames Red Enable, Not valid when ndor=1.*
- `uint32_t doy`: 1  
*Drop on Yellow enable.*
- `uint32_t cm`: 1  
*Color mode, 0b = Color blind, 1b = Color aware.*
- `uint32_t cf`: 1  
*Coupling flag, enables coupling the Committed (C) bucket and Excess (E) bucket.*
- `uint32_t ndor`: 1  
*No drop on red.*
- `netc_tb_rp_sdu_type_t sduType`: 2  
*Service Data Unit Type.*
- `uint32_t __pad0__`: 25  
*Reserved.*

### 29.5.11.2.30 struct netc\_tb\_rp\_fee\_t

#### Data Fields

- `uint8_t fen`: 1  
*Function Enable.*
- `uint8_t __pad0__`: 7  
*Reserved.*

### 29.5.11.2.31 struct netc\_tb\_rp\_pse\_t

#### Data Fields

- `uint8_t mr`: 1  
*Mark Red Flag.*
- `uint8_t res0`: 7  
*Reserved.*

### 29.5.11.2.32 struct netc\_tb\_rp\_stse\_t

#### Data Fields

- `uint32_t byteCount [2]`  
*Number of bytes received by the rate policer instance.*
- `uint32_t dropFrames [2]`  
*Number of frames dropped by the rate policer instance.*
- `uint32_t dr0GrnFrames [2]`  
*Number of frames marked green with DR=0 by the rate policer instance.*
- `uint32_t dr1GrnFrames [2]`  
*Number of frames marked green with DR=1 by the rate policer instance.*
- `uint32_t dr2GrnFrames [2]`  
*Number of frames marked yellow with DR=2 by the rate policer instance.*
- `uint32_t remarkYlwFrames [2]`

- *Number of frames re-marked from green to yellow by the rate policer instance.*
- `uint32_t dr3RedFrames [2]`  
*Number of frames marked red with DR=3 by the rate policer instance.*
- `uint32_t remarkRedFrames [2]`  
*Number of frames re-marked from green or yellow to red by the rate policer instance.*
- `uint32_t lts`  
*Last timestamp.*
- `uint32_t bci: 31`  
*Committed token bucket contents, integer portion (31 bits)*
- `uint32_t bcs: 1`  
*Committed token bucket sign bit (1 bit)*
- `uint32_t bcf`  
*Committed token bucket contents, fractional portion (31 bits)*
- `uint32_t bei`  
*Excess token bucket contents, integer portion (32 bits)*
- `uint32_t bef: 31`  
*Excess token bucket contents, fractional portion (31 bits)*
- `uint32_t bes: 1`  
*Committed token bucket sign bit.*

### 29.5.11.2.33 `struct netc_tb_rp_req_data_t`

### 29.5.11.2.34 `struct netc_tb_rp_rsp_data_t`

#### Data Fields

- `uint32_t entryID`  
*Present only for commands which perform a query.*
- `netc_tb_rp_stse_t stse`  
*Present only for commands which perform a query.*
- `netc_tb_rp_cfge_t cfge`  
*Present only for commands which perform a query.*
- `netc_tb_rp_fee_t fee`  
*Present only for commands which perform a query.*
- `netc_tb_rp_pse_t pse`  
*Present only for commands which perform a query.*

### 29.5.11.2.35 `struct netc_tb_rp_data_t`

### 29.5.11.2.36 `struct netc_tb_rp_config_t`

### 29.5.11.2.37 `struct netc_tb_isc_stse_t`

#### Data Fields

- `uint32_t rxCount`  
*Receive Count.*
- `uint32_t msduDropCount`  
*MSDU Drop Count.*

- `uint32_t policerDropCount`

*Policer Drop Count.*

- `uint32_t sgDropCount`

*Stream Gating Drop Count.*

#### 29.5.11.2.38 `struct netc_tb_isc_req_data_t`

#### 29.5.11.2.39 `struct netc_tb_isc_rsp_data_t`

#### 29.5.11.2.40 `struct netc_tb_isc_data_t`

#### 29.5.11.2.41 `struct netc_tb_sgi_cfg_t`

##### Data Fields

- `uint8_t oexen: 1`

*Octets Exceeded (Gate Closed Due To Octets Exceeded function) Enable.*

- `uint8_t irxen: 1`

*Invalid Receive (Gate Closed Due To Invalid Rx) Enable.*

- `netc_tb_sgi_sdu_type_t sduType: 2`

*The type of PDU/SDU for Interval Octets Maximum check for Gate Entry.*

#### 29.5.11.2.42 `struct netc_tb_sgi_acfge_t`

##### Data Fields

- `uint32_t adminSgclID`

*Administrative Stream Gate Control List Entry ID, 0xFFFFFFFF is NULL.*

- `uint32_t adminBaseTime [2]`

*Admin Base Time.*

- `uint32_t adminCycleTimeExt`

*Admin Cycle Time Extension.*

#### 29.5.11.2.43 `struct netc_tb_sgi_icfge_t`

##### Data Fields

- `uint8_t ipv: 4`

*Internal Priority Value (IPV), Valid if oipv is 1.*

- `uint8_t oipv: 1`

*Override frame IPV, otherwise the IPV value is determined by the stream gate control list entry.*

- `uint8_t gst: 1`

*Specifies Gate State before the administrative stream gate control list takes affect, 0b = Closed; 1b = Open.*

- `uint8_t ctd: 1`

*Specifies Cut Through disable status before the administrative stream gate control list takes affect , Not applicable to ENETC function.*

### 29.5.11.2.44 struct netc\_tb\_sgi\_sgise\_t

#### Data Fields

- uint32\_t **operSgclEID**  
*Operational Stream Gate Control List Entry ID.*
- uint32\_t **configChangeTime** [2]  
*Configuration Change Time.*
- uint32\_t **operBaseTime** [2]  
*Operational Base Time.*
- uint32\_t **operCycleTimeExt**  
*Oper Cycle Time Extension.*
- uint32\_t **oex**: 1  
*Octets Exceeded Flag.*
- uint32\_t **irx**: 1  
*Invalid Receive Flag.*
- **netc\_tb\_sgi\_state\_t state**: 3  
*Current Gate Instance State.*

### 29.5.11.2.45 struct netc\_tb\_sgi\_req\_data\_t

### 29.5.11.2.46 struct netc\_tb\_sgi\_rsp\_data\_t

### 29.5.11.2.47 struct netc\_tb\_sgi\_data\_t

### 29.5.11.2.48 struct netc\_tb\_sgi\_config\_t

### 29.5.11.2.49 struct netc\_sgcl\_gate\_entry\_t

#### Data Fields

- uint32\_t **timeInterval**  
*Time Interval for Gate Entry.*
- uint32\_t **iom**: 24  
*Interval Octets Maximum for Gate Entry, specifies the maximum bytes (octets) allowed to pass (open), valid if iomen = 1.*
- uint32\_t **ipv**: 4  
*Internal Priority Value for Gate Entry.*
- uint32\_t **oipv**: 1  
*Override Internal Priority Value for Gate Entry.*
- uint32\_t **ctd**: 1  
*Cut Through Disable for Gate Entry.*
- uint32\_t **iomen**: 1  
*Interval Octet Maximum Enabled for Gate Entry, 0b = Don'ttrack count, 1b = Track count.*
- uint32\_t **gtst**: 1  
*Gate State for Gate Entry, 0b = Closed; 1b = Open.*

### 29.5.11.2.50 struct netc\_tb\_sgcl\_cfge\_t

#### Data Fields

- uint32\_t **cycleTime**  
*Cycle Time.*
- uint8\_t **listLength**  
*List Length.*
- uint16\_t **extOipv**: 1  
*Extension (means the stream gate control list ends and before cycleTime restarts) Override Internal Priority Value.*
- uint16\_t **extIpv**: 4  
*List Extension Internal Priority Value, valid if extOipv = 1.*
- uint16\_t **extCtd**: 1  
*Extension Cut Through Disabled, 0b = No action, 1b = Disabled.*
- uint16\_t **extGtst**: 1  
*Extension Gate State, 0b = closed, 1b = Open.*

### 29.5.11.2.51 struct netc\_tb\_sgcl\_sgclse\_t

#### Data Fields

- uint8\_t **refCount**  
*Reference Count, 1 indicates that the gate control list is an administrative or an operational gate control list in a stream gate instance.*

### 29.5.11.2.52 struct netc\_tb\_sgcl\_req\_data\_t

### 29.5.11.2.53 struct netc\_tb\_sgcl\_rsp\_data\_t

### 29.5.11.2.54 struct netc\_tb\_sgcl\_data\_t

### 29.5.11.2.55 struct netc\_tb\_sgcl\_gcl\_t

#### Data Fields

- uint16\_t **extOipv**: 1  
*Extension (means the stream gate control list ends and before cycleTime restarts) Override Internal Priority Value.*
- uint16\_t **extIpv**: 4  
*List Extension Internal Priority Value, valid if extOipv = 1.*
- uint16\_t **extCtd**: 1  
*Extension Cut Through Disabled, 0b = No action, 1b = Disabled.*
- uint16\_t **extGtst**: 1  
*Extension Gate State, 0b = closed, 1b = Open.*
- uint32\_t **cycleTime**  
*Cycle Time.*
- uint32\_t **numEntries**

- `netc_sgcl_gate_entry_t * gcList`  
*Control List entry numbers.  
 Pointer to stream gate control list array.*

### 29.5.11.2.56 struct netc\_tb\_fm\_cfge\_t

#### Data Fields

- `netc_tb_fm_layer2_act_t l2Act: 1`  
*Layer 2 Actions.*
- `netc_tb_fm_mac_header_act_t macHdrAct: 3`  
*Layer 2 Header MAC Actions.*
- `netc_tb_fm_vlan_header_act_t vlanHdrAct: 2`  
*Layer 2 VLAN Actions.*
- `netc_tb_fm_outer_vid_act_t outerVidAct: 2`  
*Outer VID Actions.*
- `netc_tb_fm_sqt_act_t sqtAct: 3`  
*Sequence Tag Action, Not applicable for ingress frame modifications.*
- `uint16_t smacPort: 5`  
*Source MAC Address Register Port, valid if macHdrAct=010b,011b,100b.*
- `uint8_t dmac [6]`  
*Destination MAC Address, valid if macHdrAct = 011b,101b.*
- `uint32_t outerVlanID: 12`  
*Outer VLAN VID, valid if outerVidAct = 01b.*
- `uint32_t outerVlanPcp: 3`  
*Outer VLAN PCP, valid if outerPcpAct = 01b.*
- `uint32_t outerVlanDei: 1`  
*Outer VLAN DEI, valid if outerDeiAct = 01b.*
- `netc_tb_fm_outer_tpid_act_t outerTpidAct: 3`  
*Outer TPID action.*
- `netc_tb_fm_outer_pcp_act_t outerPcpAct: 3`  
*Outer PCP action.*
- `netc_tb_fm_outer_dei_act_t outerDeiAct: 2`  
*Outer DEI action.*
- `netc_tb_fm_payload_act_t pldAct: 3`  
*Payload Actions, Not applicable for ingress frame modifications.*
- `uint8_t pldOffset`  
*Payload Offset, valid if outerPldAct = 010b.*
- `uint16_t fmdBytes`  
*Frame Modification Bytes, valid if outerPldAct = 001b,010b or l2Act = 1b.*
- `uint32_t fmdEID`  
*Frame Modification Data Entry ID, valid if outerPldAct = 001b,010b or l2Act = 1b.*

#### Field Documentation

##### (1) `uint32_t netc_tb_fm_cfge_t::fmdEID`

0xFFFF is null pointer

**29.5.11.2.57 struct netc\_tb\_fm\_req\_data\_t**

**29.5.11.2.58 struct netc\_tb\_fm\_rsp\_data\_t**

**29.5.11.2.59 struct netc\_tb\_fm\_data\_t**

**29.5.11.2.60 struct netc\_tb\_fm\_config\_t**

**29.5.11.2.61 struct netc\_tb\_fmd\_req\_data\_t**

#### Data Fields

- uint8\_t **cfg[e]** []

*Configuration Element Data size is variable.*

**29.5.11.2.62 struct netc\_tb\_fmd\_rsp\_data\_t**

#### Data Fields

- uint8\_t **cfg[e]** []

*Configuration Element Data size is variable.*

**29.5.11.2.63 struct netc\_tb\_fmd\_data\_t**

**29.5.11.2.64 struct netc\_tb\_fmd\_update\_config\_t**

#### Data Fields

- uint32\_t **res**

*Hold for request->commonHeader.*

- uint8\_t **cfg[e]** []

*Configuration Element Data size is variable.*

**29.5.11.2.65 struct netc\_tb\_fmd\_query\_buffer\_t**

#### Data Fields

- uint32\_t **entryID**

*EntryID of the queried entry.*

- uint8\_t **cfg[e]** []

*Configuration Element Data size is variable.*

**29.5.11.2.66 struct netc\_tb\_vf\_keye\_t****29.5.11.2.67 struct netc\_tb\_vf\_cfge\_t****Data Fields**

- uint32\_t **portMembership**: 24  
*Port Membership Bitmap.*
- uint32\_t **stgID**: 4  
*Spanning Tree Group Member ID.*
- uint32\_t **fid**: 12  
*Filtering ID.*
- uint32\_t **mlo**: 3  
*MAC Learning Options.*
- uint32\_t **mfo**: 2  
*MAC Forwarding Options.*
- uint32\_t **ipmfe**: 1  
*IP Multicast Filtering Enable.*
- uint32\_t **ipmfle**: 1  
*IP Multicast Flooding Enable.*
- uint32\_t **etaPortBitmap**: 24  
*Egress Treatment Applicability Port Bitmap for the secondary Egress Treatment group.*
- uint32\_t **baseETEID**  
*Base Egress Treatment Entry ID for the secondary Egress Treatment group.*

**29.5.11.2.68 struct netc\_tb\_vf\_search\_criteria\_t****Data Fields**

- uint32\_t **resumeEntryId**  
*Resume Entry ID, when starting a search, pass the NULL Entry ID.*

**Field Documentation****(1) uint32\_t netc\_tb\_vf\_search\_criteria\_t::resumeEntryId****29.5.11.2.69 struct netc\_tb\_vf\_req\_data\_t****Data Fields**

- **netc\_tb\_vf\_cfge\_t cfge**  
*Present only for update or add commands.*
- **netc\_tb\_vf\_search\_criteria\_t sCriteria**  
*Active when access method is kNETC\_Search.*

### 29.5.11.2.70 struct netc\_tb\_vf\_rsp\_data\_t

#### Data Fields

- `uint32_t status`  
*Present only for query command with search access method.*
- `uint32_t entryID`  
*Present only for query command.*
- `netc_tb_vf_keye_t keye`  
*Present only for query command.*
- `netc_tb_vf_cfge_t cfge`  
*Present only for query command.*

### 29.5.11.2.71 struct netc\_tb\_vf\_data\_t

### 29.5.11.2.72 struct netc\_tb\_vf\_config\_t

### 29.5.11.2.73 struct netc\_tb\_fdb\_cfge\_t

#### Data Fields

- `uint32_t portBitmap: 24`  
*Forwarding destination Port Bitmap and ET applicability port bitmap when oETEID = 10b.*
- `netc_tb_fdb_oeteid_mode_t oETEID: 2`  
*Override ET\_EID option.*
- `uint32_t ePort: 5`  
*Egress Ports, active when oETEid = 01b or ctd = 01b.*
- `uint32_t iMirE: 1`  
*Ingress Mirroring Enable.*
- `netc_tb_fdb_ctd_mode_t ctd: 2`  
*Cut-Through Disable.*
- `uint32_t dynamic: 1`  
*Static or Dynamic Entry, 0b = Static entry, 1b = Dynamic entry.*
- `uint32_t timeCapE: 1`  
*Timestamp Capture Enable when set.*
- `uint32_t etEID`  
*Base egress treatment table entry id for primary Egress Treatment group, is valid if the oETEID field is set to value other than kNETC\_FDBNoEPP.*

#### Field Documentation

##### (1) `uint32_t netc_tb_fdb_cfge_t::etEID`

0xFFFFFFFF is NULL.

### 29.5.11.2.74 struct netc\_tb\_fdb\_acte\_t

#### Data Fields

- uint8\_t **actCnt**: 7  
*Activity Counter.*
- uint8\_t **actFlag**: 1  
*Activity Flag.*

### 29.5.11.2.75 struct netc\_tb\_fdb\_search\_criteria\_t

#### Data Fields

- uint32\_t **resumeEntryId**  
*Resume Entry ID, pass the NULL Entry ID when starting a search.*
- netc\_tb\_fdb\_keye\_t **keye**  
*Key Element data which used to match against the table entries.*
- netc\_tb\_cfge\_t **cfge**  
*Configuration Element data which used to match against the table entries.*
- netc\_tb\_fdb\_acte\_t **acte**  
*Activity Element data which used to match against the table entries.*
- netc\_tb\_fdb\_sc\_keye\_mc\_t **keyeMc**: 2  
*Key Element data match criteria.*
- netc\_tb\_fdb\_sc\_cfge\_mc\_t **cfgeMc**: 3  
*Configuration Element data match criteria.*
- netc\_tb\_fdb\_sc\_acte\_mc\_t **acteMc**: 1  
*Activity Element data match criteria.*

### 29.5.11.2.76 struct netc\_tb\_fdb\_req\_data\_t

#### Data Fields

- netc\_tb\_common\_header\_t **commonHeader**  
*Define update actions (use netc\_tb\_fdb\_update\_action\_t) and query actions.*
- netc\_tb\_cfge\_t **cfge**  
*Present only for commands which perform an update or add.*
- uint32\_t **entryID**  
*Active when access method is kNETC\_EntryIDMatch.*
- netc\_tb\_fdb\_keye\_t **keye**  
*Active when access method is kNETC\_ExactKeyMatch.*
- netc\_tb\_fdb\_search\_criteria\_t **sCriteria**  
*Active when access method is kNETC\_Search.*

### 29.5.11.2.77 struct netc\_tb\_fdb\_rsp\_data\_t

#### Data Fields

- uint32\_t **status**

*RESUME\_ENTRY\_ID, valid only in responses for commands which use the Search Access Method.*

- `uint32_t entryID`  
*Present only for query command.*
- `netc_tb_fdb_keye_t keye`  
*Present only for query command.*
- `netc_tb_fdb_cfge_t cfge`  
*Present only for query command.*
- `netc_tb_fdb_acte_t acte`  
*Present only for query command.*

#### 29.5.11.2.78 `struct netc_tb_fdb_data_t`

#### 29.5.11.2.79 `struct netc_tb_fdb_config_t`

#### 29.5.11.2.80 `struct netc_tb_l2mcf_keye_t`

##### Data Fields

- `netc_tb_l2mcf_key_type_t keyType: 4`  
*Key Type.*
- `uint32_t fid: 12`  
*Filtering ID.*
- `uint32_t ipv4DestAddr`  
*IPv4 Destination Address.*
- `uint32_t ipv4SrcAddr`  
*IPv4 Source Address.*

#### 29.5.11.2.81 `struct netc_tb_l2mcf_search_criteria_t`

##### Data Fields

- `uint32_t resumeEntryId`  
*Resume Entry ID, pass the NULL Entry ID when starting a search.*
- `netc_tb_l2mcf_keye_t keye`  
*Key Element data which used to match against the table entries.*
- `netc_tb_l2mcf_cfge_t cfge`  
*Configuration Element data which used to match against the table entries.*
- `netc_tb_l2mcf_acte_t acte`  
*Activity Element data which used to match against the table entries.*
- `etc_tb_l2mcf_sc_keye_mc_t keyeMc: 1`  
*Key Element data match criteria.*
- `etc_tb_l2mcf_sc_cfge_mc_t cfgeMc: 3`  
*Configuration Element data match criteria.*
- `etc_tb_l2mcf_sc_acte_mc_t acteMc: 1`  
*Activity Element data match criteria.*

**29.5.11.2.82 struct netc\_tb\_l2mcf\_req\_data\_t**

**29.5.11.2.83 struct netc\_tb\_l2mcf\_rsp\_data\_t**

**29.5.11.2.84 struct netc\_tb\_l2mcf\_data\_t**

**29.5.11.2.85 struct netc\_tb\_l2mcf\_config\_t**

**29.5.11.2.86 struct netc\_tb\_isseqg\_cfge\_t**

#### Data Fields

- **netc\_tb\_isseqg\_sqtag\_t sqTag:** 3  
*Sequence Tag Type.*
- **uint8\_t \_\_pad0\_\_:** 5  
*Reserved.*

#### Field Documentation

(1) **netc\_tb\_isseqg\_sqtag\_t netc\_tb\_isseqg\_cfge\_t::sqTag**

(2) **uint8\_t netc\_tb\_isseqg\_cfge\_t::\_\_pad0\_\_**

**29.5.11.2.87 struct netc\_tb\_isseqg\_sgse\_t**

#### Data Fields

- **uint16\_t sqgNum**  
*Sequence Generation Number.*

**29.5.11.2.88 struct netc\_tb\_isseqg\_req\_data\_t**

**29.5.11.2.89 struct netc\_tb\_isseqg\_rsp\_data\_t**

**29.5.11.2.90 struct netc\_tb\_isseqg\_data\_t**

**29.5.11.2.91 struct netc\_tb\_isseqg\_config\_t**

**29.5.11.2.92 struct netc\_tb\_eseqr\_cfge\_t**

#### Data Fields

- **netc\_tb\_eseqr\_sqtag\_t sqTag:** 3  
*Sequence Tag, specify the expected sequence tag type in the frame.*
- **uint32\_t sqrTnsq:** 1  
*Sequence Recovery Take No Sequence.*
- **uint32\_t sqrAlg:** 1  
*Sequence Recovery Algorithm, 0b = Vector algorithm, 1b = Match algorithm.*
- **uint32\_t sqrType:** 1

- **Sequence Recovery Function type, 0b = Sequence recovery function, 1b = Individual recovery function.**
- **uint32\_t sqrHl:** 7  
*Sequence Recovery History Length, valid if sqrAlg = 0b.*
- **uint32\_t sqrFwl:** 12  
*Sequence Recovery Future Window Length, valid if sqrAlg = 0b.*
- **uint32\_t sqrTp:** 12  
*Sequence Timeout Period, the unit is 1.048576 milliseconds.*

### 29.5.11.2.93 struct netc\_tb\_esqr\_stse\_t

#### Data Fields

- **uint32\_t inOrderPackets [2]**  
*In Order Packets.*
- **uint32\_t outOfOrderPackets [2]**  
*Out of Order Packets.*
- **uint32\_t roguePackets [2]**  
*Rogue Packets.*
- **uint32\_t duplicatePackets [2]**  
*Duplicate Packets.*
- **uint32\_t lostPackets [2]**  
*Lost Packets.*
- **uint32\_t taglessPackets [2]**  
*Tag-Less Packets.*
- **uint32\_t esqrResetCounts**  
*Sequence Recovery Resets.*

### 29.5.11.2.94 struct netc\_tb\_esqr\_srse\_t

#### Data Fields

- **uint32\_t sqrNum:** 16  
*Sequence Recovery Number.*
- **uint32\_t takeAny:** 1  
*Take Any.*
- **uint32\_t lce:** 1  
*Lost Count Enable.*
- **uint32\_t sqrTs:** 12  
*Sequence Recovery Timestamp.*
- **uint32\_t sqrHistory [4]**  
*Recovery History bit vector, each bit corresponding to sequence numbers, bit 1 means a packet with that sequence number has been previously received.*

**29.5.11.2.95 struct netc\_tb\_eseqr\_req\_data\_t**

**29.5.11.2.96 struct netc\_tb\_eseqr\_rsp\_data\_t**

**29.5.11.2.97 struct netc\_tb\_eseqr\_data\_t**

**29.5.11.2.98 struct netc\_tb\_eseqr\_config\_t**

**29.5.11.2.99 struct netc\_tgs\_gate\_entry\_t**

#### Data Fields

- **uint32\_t interval**  
*Entry Time Interval.*
- **uint32\_t tcGateState:** 8  
*Traffic Class Gate States for Gate Entry, 8 bits for 8 Traffic Class , 0b means Gate closed, 1b means Gate open.*
- **netc\_tb\_tgs\_gate\_type\_t operType:** 4  
*Gate operation type ( IEEE 802.1Q-2018) field for gate control list entry i.*
- **uint32\_t gate**  
*Entry Gate Mask.*

**29.5.11.2.100 struct netc\_tb\_tgs\_cfge\_t**

#### Data Fields

- **uint64\_t adminBaseTime**  
*Administrative Base Time.*
- **uint32\_t adminCycleTime**  
*Administrative Cycle Time.*
- **uint32\_t adminCycleTimeExt**  
*Administrative Cycle Time Extension.*
- **uint32\_t adminControlListLength**  
*Administrative Control List Length.*
- **netc\_tgs\_gate\_entry\_t adminGcl []**  
*Administrative Gate control list.*

**29.5.11.2.101 struct netc\_tb\_tgs\_olse\_t**

#### Data Fields

- **uint64\_t configChangeTime**  
*The time at which this operational gate control list became active.*
- **uint64\_t configChangeError**  
*Count of error configuration changes.*
- **uint64\_t operBaseTime**  
*Operational Base Time.*
- **uint32\_t operCycleTime**

- *Operational Cycle Time.*
- `uint32_t operCycleTimeExt`  
*Operational Cycle Time Extension.*
- `uint32_t operControlListLength`  
*Operational Control List Length.*
- `netc_tgs_gate_entry_t operGcl []`  
*Operational Gate control list.*

### 29.5.11.2.102 struct netc\_tb\_tgs\_req\_data\_t

#### Data Fields

- `netc_tb_tgs_cfge_t cfge`  
*Present only for commands which perform a update.*

### 29.5.11.2.103 struct netc\_tb\_tgs\_rsp\_data\_t

#### Data Fields

- `uint32_t entryID`  
*Present only for commands which perform a query.*
- `netc_tb_tgs_cfge_t cfge`  
*Present only for commands which perform a query.*
- `netc_tb_tgs_olse_t olse`  
*Present only for commands which perform a query.*

### 29.5.11.2.104 struct netc\_tb\_tgs\_data\_t

### 29.5.11.2.105 struct netc\_tb\_tgs\_gcl\_t

#### Data Fields

- `uint64_t baseTime`  
*Base Time.*
- `uint32_t cycleTime`  
*Cycle Time.*
- `uint32_t extTime`  
*Cycle Time Extension.*
- `uint32_t numEntries`  
*Control List entry numbers.*
- `netc_tgs_gate_entry_t * gcList`  
*Pointer to time gate control list array.*

### 29.5.11.2.106 struct netc\_tb\_et\_cfge\_t

#### Data Fields

- `netc_tb_et_efm_mode_t efmMode`: 2  
*Egress Frame Modification mode.*
- `netc_tb_et_esq_act_t esqa`: 2  
*Egress Sequence Actions.*
- `netc_tb_et_ec_act_t eca`: 2  
*Egress Counter Action.*
- `uint8_t __pad1__`: 1  
*Reserve for data align.*
- `uint8_t efmLenChange`: 7  
*Egress Frame Modification Length Change, specified in units of bytes using a 2's complement notation.*
- `uint16_t efmDataLen`: 11  
*Egress Frame Modification Data Length.*
- `uint32_t efmEID`  
*Egress Frame Modification Entry Id.*
- `uint32_t ecEID`  
*Egress Count Table Entry ID.*
- `uint32_t esqaTgtEID`  
*Egress Sequence Actions Target Entry ID, active when esqa = 10b.*

### 29.5.11.2.107 struct netc\_tb\_et\_req\_data\_t

### 29.5.11.2.108 struct netc\_tb\_et\_rsp\_data\_t

### 29.5.11.2.109 struct netc\_tb\_et\_data\_t

### 29.5.11.2.110 struct netc\_tb\_et\_config\_t

### 29.5.11.2.111 struct netc\_tb\_etmcq\_cfge\_t

#### Data Fields

- `netc_hw_etm_class_queue_idx_t cq2cgMap`: 4  
*Class Queue to Congestion Group Mapping.*

### 29.5.11.2.112 struct netc\_tb\_etmcq\_stse\_t

#### Data Fields

- `uint32_t rejByteCnt [2]`  
*Reject Byte Count.*
- `uint32_t rejFrameCnt [2]`  
*Reject Frame Count.*
- `uint32_t deqByteCnt [2]`  
*Dequeue Byte Count.*

- `uint32_t deqFrameCnt [2]`  
*Dequeue Frame Count.*
- `uint32_t dropByteCnt [2]`  
*Dropped Frames, Memory Lost.*
- `uint32_t dropFrameCnt [2]`  
*Dropped Frames, Memory Recovered.*
- `uint32_t frmCnt`  
*Frame Count.*

**29.5.11.2.113 struct netc\_tb\_etmcq\_req\_data\_t**

**29.5.11.2.114 struct netc\_tb\_etmcq\_rsp\_data\_t**

**29.5.11.2.115 struct netc\_tb\_etmcq\_data\_t**

**29.5.11.2.116 struct netc\_tb\_etmcq\_config\_t**

#### Data Fields

- `uint32_t entryID`  
*Need use NETC\_TB\_ETM\_CQ\_ENTRY\_ID macro to create entry ID.*

**29.5.11.2.117 struct netc\_tb\_etmcs\_cfge\_t**

#### Data Fields

- `netc_tb_etmcs_ca_assg_t cqAssg: 4`  
*Class Queue Assignment, input 0 to 7 are weighted fair whereby input 8 to 15 are strict priority.*
- `uint32_t oal: 11`  
*Overread accounting length.*
- `struct {`
  - `uint8_t xCode: 3`  
*Weight code x value.*
  - `uint8_t yCode: 5`  
*Weight code y value.*
- `} wbfsWeight [8]`

*Weight for scheduler input 0 ~ 7, effective weight is:  $(2^x)/(1-(y/64))$*

**29.5.11.2.118 struct netc\_tb\_etmcs\_req\_data\_t**

#### Data Fields

- `netc_tb_etmcs_entry_id_t entryID`  
*One class scheduler entry per port.*

**29.5.11.2.119 struct netc\_tb\_etmcs\_rsp\_data\_t**

**29.5.11.2.120 struct netc\_tb\_etmcs\_data\_t**

**29.5.11.2.121 struct netc\_tb\_etmcs\_config\_t**

#### Data Fields

- **netc\_tb\_etmcs\_entry\_id\_t entryID**  
*One class scheduler entry per port.*

**29.5.11.2.122 struct netc\_tb\_etmcg\_cfge\_t**

#### Data Fields

- **uint16\_t tdDr0En:** 1  
*Tail drop enable for DR0 Frame.*
- **uint16\_t tdDr1En:** 1  
*Tail drop enable for DR1 Frame.*
- **uint16\_t tdDr2En:** 1  
*Tail drop enable for DR2 Frame.*
- **uint16\_t tdDr3En:** 1  
*Tail drop enable for DR3 Frame.*
- **uint16\_t oal:** 12  
*Overhead accounting length, 2's complement value (range -2048 to +2047)*
- **struct {**
  - uint16\_t tn:** 5  
*TA.*
  - uint16\_t ta:** 8  
*Tn.***}** **tdDRThresh [4]**

*Tail Drop Threshold ( $TA * 2^Tn$ ) for DR0 ~ DR3 Frames, valid if  $tdDrnEn = 1b$ .*

**29.5.11.2.123 struct netc\_tb\_etmcg\_stse\_t**

#### Data Fields

- **uint32\_t byteCount [2]**  
*Number of bytes currently in use in all class queues that are members of this group.*

**Field Documentation**

(1) `uint32_t netc_tb_etmcg_stse_t::byteCount[2]`

**29.5.11.2.124 struct netc\_tb\_etmcg\_req\_data\_t**

**29.5.11.2.125 struct netc\_tb\_etmcg\_rsp\_data\_t**

**29.5.11.2.126 struct netc\_tb\_etmcg\_data\_t**

**29.5.11.2.127 struct netc\_tb\_etmcg\_config\_t**

**Data Fields**

- `uint32_t entryID`  
*Need use NETC\_TB\_ETM(CG)\_ENTRY\_ID macro to create entry ID.*

**29.5.11.2.128 struct netc\_tb\_ec\_stse\_t**

**Data Fields**

- `uint32_t enqFrmCnt [2]`  
*Enqueued Frame Count.*
- `uint32_t rejFrmCnt [2]`  
*Rejected Frame Count.*

**29.5.11.2.129 struct netc\_tb\_ec\_req\_data\_t**

**29.5.11.2.130 struct netc\_tb\_ec\_rsp\_data\_t**

**29.5.11.2.131 struct netc\_tb\_ec\_data\_t**

**29.5.11.2.132 struct netc\_tb\_bp\_cfge\_t**

**Data Fields**

- `bool sbpEn: 1`  
*Shared Buffer Pool Enable, set true means a shared buffer pool is associated with this buffer pool.*
- `netc_tb_bp_fc_cfg_t gcCfg: 2`  
*Flow Control (FC) Configuration.*
- `uint8_t pfcVector`  
*Priority Flow Control (PFC) Vector, not support in NETC 3.0 and 3.1 version.*
- `uint16_t maxThresh: 12`  
*Maximum Threshold, value 0 means disable maximum threshold checking, use NETC\_TB\_BP\_THRESH macro to set this value.*
- `uint16_t fcOnThresh: 12`

*Flow Control On Threshold, If the buffer pool usage crosses this threshold, and if fcOnThresh is greater than fcOffThresh, the flow control state of the buffer pool is set to 1, use NETC\_TB\_BP\_THRESH macro to set this value.*

- **uint16\_t fcOffThresh:** 12  
*Flow Control Off Threshold, If buffer pool usage drops to this threshold or below, the flow control state of the buffer pool is set to 0, , use NETC\_TB\_BP\_THRESH macro to set this value.*
- **uint32\_t sbpThresh:** 12  
*Shared Buffer Pool Threshold, use NETC\_TB\_BP\_THRESH macro to set this value.*
- **uint32\_t sbpEid**  
*Shared Buffer Pool Entry ID, valid if sbpEn is true.*
- **uint32\_t fcPorts**  
*Flow Control Port bitmap, indicates which ports are to be flow controlled for this buffer pool.*

## Field Documentation

(1) **uint16\_t netc\_tb\_bp\_cfge\_t::fcOnThresh**

**29.5.11.2.133 struct netc\_tb\_bp\_bpse\_t**

### Data Fields

- **uint32\_t amountUsed**  
*Amount Used, number of internal memory words (average of 20 bytes each) currently in use in this buffer pool.*
- **uint32\_t amountUsedHWM**  
*Amount Used High Watermark, value sticks at the highest AMOUNT\_USED seen since the last watermark reset.*
- **uint32\_t fcState:** 1  
*Flow Control (FC) State, ON (1) or OFF (0)*
- **uint32\_t bpd:** 1  
*Buffer Pool Disabled, 1 means the buffer pool has been disabled due to an uncorrectable ECC error.*

## Field Documentation

(1) **uint32\_t netc\_tb\_bp\_bpse\_t::amountUsed**

**29.5.11.2.134 struct netc\_tb\_bp\_req\_data\_t**

**29.5.11.2.135 struct netc\_tb\_bp\_rsp\_data\_t**

**29.5.11.2.136 struct netc\_tb\_bp\_data\_t**

**29.5.11.2.137 struct netc\_tb\_bp\_config\_t**

### Data Fields

- **uint32\_t entryID**  
*Buffer pool ID, range in 0 ~ (SWT\_GetBPTableEntryNum() - 1)*
- **netc\_tb\_bp\_cfge\_t cfge**  
*Buffer Pool table config element.*

**29.5.11.2.138 struct netc\_tb\_sbp\_cfge\_t****Data Fields**

- uint32\_t **maxThresh**: 12  
*Maximum Threshold, If shared buffer pool usage is greater than or equal to this threshold, use NETC\_TB\_BP\_THRESH macro to set this value.*
- uint16\_t **fcOnThresh**: 12  
*Flow Control On Threshold, If the shared buffer pool usage crosses this threshold, and if fcOnThresh is greater than fcOffThresh, the flow control state of the buffer pool is set to 1, use NETC\_TB\_BP\_THRESH macro to set this value.*
- uint16\_t **fcOffThresh**: 12  
*Flow Control Off Threshold, If shared buffer pool usage drops to this threshold or below, the flow control state of the buffer pool is set to 0, use NETC\_TB\_BP\_THRESH macro to set this value.*

**Field Documentation**(1) **uint16\_t netc\_tb\_sbp\_cfge\_t::fcOnThresh****29.5.11.2.139 struct netc\_tb\_sbp\_sbpsc\_t****Data Fields**

- uint32\_t **amountUsed**  
*Amount Used, number of internal memory words (average of 20 bytes each) currently in use in this buffer pool.*
- uint32\_t **amountUsedHWM**  
*Amount Used High Watermark, value sticks at the highest AMOUNT\_USED seen since the last watermark reset.*
- uint32\_t **fcState**: 1  
*Flow Control (FC) State, ON (1) or OFF (0)*

**Field Documentation**(1) **uint32\_t netc\_tb\_sbp\_sbpsc\_t::amountUsed****29.5.11.2.140 struct netc\_tb\_sbp\_req\_data\_t****29.5.11.2.141 struct netc\_tb\_sbp\_rsp\_data\_t****29.5.11.2.142 struct netc\_tb\_sbp\_data\_t****29.5.11.2.143 struct netc\_tb\_sbp\_config\_t****Data Fields**

- uint32\_t **entryID**  
*Shared Buffer pool ID, range in 0 ~ (SWT\_GetSBPTableEntryNum() - 1)*
- **netc\_tb\_sbp\_cfge\_t cfge**  
*Shared Buffer Pool table config element.*

### 29.5.11.2.144 union netc\_tb\_data\_buffer\_t

#### Data Fields

- [netc\\_tb\\_tgs\\_data\\_t tgs](#)  
*Time Gate Scheduling table data buffer.*
- [netc\\_tb\\_rp\\_data\\_t rp](#)  
*Rate Policer table data buffer.*
- [netc\\_tb\\_ipf\\_data\\_t ipf](#)  
*Ingress Port filter table data buffer.*
- [netc\\_tb\\_fdb\\_data\\_t fdb](#)  
*FDB table data buffer.*
- [netc\\_tb\\_l2mcf\\_data\\_t l2mcf](#)  
*L2 IPV4 Multicast Filter table data buffer.*
- [netc\\_tb\\_vf\\_data\\_t vf](#)  
*VLAN Filter table data buffer.*
- [netc\\_tb\\_isi\\_data\\_t isi](#)  
*Ingress Stream Identification table data buffer.*
- [netc\\_tb\\_is\\_data\\_t is](#)  
*Ingress Stream table data buffer.*
- [netc\\_tb\\_isf\\_data\\_t isf](#)  
*Ingress Stream Filter table data buffer.*
- [netc\\_tb\\_isc\\_data\\_t isc](#)  
*Ingress Stream Count table data buffer.*
- [netc\\_tb\\_sgi\\_data\\_t sgi](#)  
*Stream Gate Instance table data buffer.*
- [netc\\_tb\\_sgcl\\_data\\_t sgcl](#)  
*Stream Gate Control List table data buffer.*
- [netc\\_tb\\_fm\\_data\\_t fm](#)  
*Frame Modification table data buffer.*
- [netc\\_tb\\_fmd\\_data\\_t fmd](#)  
*Frame Modification Data table data buffer.*
- [netc\\_tb\\_et\\_data\\_t et](#)  
*Egress Treatment table data buffer.*
- [netc\\_tb\\_ec\\_data\\_t ec](#)  
*Egress Count table data buffer.*
- [netc\\_tb\\_etmcq\\_data\\_t cq](#)  
*ETM Class Queue table data buffer.*
- [netc\\_tb\\_etmcs\\_data\\_t cs](#)  
*ETM Class Scheduler table data buffer.*
- [netc\\_tb\\_etmcg\\_data\\_t cg](#)  
*ETM Class Group table data buffer.*
- [netc\\_tb\\_iseqg\\_data\\_t iseqg](#)  
*Ingress Sequence Generation table data buffer.*
- [netc\\_tb\\_eseqr\\_data\\_t eseqr](#)  
*Egress Sequence Recovery table data buffer.*
- [netc\\_tb\\_bp\\_data\\_t bp](#)  
*Buffer Pool table data buffer.*
- [netc\\_tb\\_sbp\\_data\\_t sbp](#)  
*Shared Buffer Pool table data buffer.*

### 29.5.11.2.145 struct netc\_cbdr\_hw\_t

#### Data Fields

- `__IO uint32_t CBDRMR`  
*Command BDR mode register.*
- `__I uint32_t CBDRSR`  
*Command BDR status register.*
- `__IO uint32_t CBDRBAR0`  
*Command BDR base address register 0.*
- `__IO uint32_t CBDRBAR1`  
*Command BDR base address register 1.*
- `__IO uint32_t CBDRPIR`  
*Command BDR producer index register.*
- `__IO uint32_t CBDRCIR`  
*Command BDR consumer index register.*
- `__IO uint32_t CBDRLENR`  
*Command BDR length register.*

#### Field Documentation

- (1) `__IO uint32_t netc_cbdr_hw_t::CBDRMR`
- (2) `__I uint32_t netc_cbdr_hw_t::CBDRSR`

### 29.5.11.2.146 struct netc\_cbdr\_handle\_t

#### Data Fields

- `netc_cbdr_hw_t * base`  
*Point to hardware command bd ring register group.*
- `netc_cmd_bdr_t * cmdr`  
*Point to command BD ring handle.*
- `netc_tb_data_buffer_t * buffer`  
*Point to table common data buffer.*

#### Field Documentation

- (1) `netc_cbdr_hw_t* netc_cbdr_handle_t::base`
- (2) `netc_cmd_bdr_t* netc_cbdr_handle_t::cmdr`
- (3) `netc_tb_data_buffer_t* netc_cbdr_handle_t::buffer`

### 29.5.11.3 Macro Definition Documentation

#### 29.5.11.3.1 #define NETC\_FD\_EID\_ENCODE\_OPTION\_0( *entryId* ) ((uint32\_t)(*entryId*)&0xFFFFU)

Note

sqta should be netc\_fm\_sqt\_act\_t type, vuda should be netc\_fm\_vlan\_ud\_act\_t type and vara should be netc\_fm\_vlan\_ar\_act\_t type.

#### 29.5.11.4 Typedef Documentation

##### 29.5.11.4.1 `typedef netc_fm_sqt_act_t netc_tb_fm_sqt_act_t`

Note

For use kNETC\_ReomveTag action, If R-TAG/draft 2.0 R-TAG/HSR tag not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

#### 29.5.11.5 Enumeration Type Documentation

##### 29.5.11.5.1 `enum netc_tb_index_t`

Enumerator

- kNETC\_TGSTable* Time Gate Scheduling table index.
- kNETC\_RPTable* Rate Policer table index.
- kNETC\_IPFTable* Ingress Port filter table index.
- kNETC\_FDBTable* FDB table index.
- kNETC\_L2MCFTable* L2 IPV4 Multicast Filter table index.
- kNETC\_VFTable* VLAN Filter table index.
- kNETC\_ECQTable* ETM Class Queue table index.
- kNETC\_ECSTable* ETM Class Scheduler table index.
- kNETC\_ISITable* Ingress Stream Identification table index.
- kNETC\_ISTable* Ingress Stream table index.
- kNETC\_ISFTable* Ingress Stream Filter table index.
- kNETC\_ETTable* Egress Treatment table index.
- kNETC\_ISGTable* Ingress Sequence Generation table index.
- kNETC\_ESRTable* Egress Sequence Recovery table index.
- kNETC\_SGITable* Stream Gate Instance table index.
- kNETC\_SGCLTable* Stream Gate Control List table index.
- kNETC\_ISCTable* Ingress Stream Count table index.
- kNETC\_ECTable* Egress Count table index.
- kNETC\_FMTTable* Frame Modification table index.
- kNETC\_BPTable* Buffer Pool table index.
- kNETC\_SBPTable* Shared Buffer Pool table index.
- kNETC\_ECGTable* ETM Class Group table index.
- kNETC\_FMDataTable* Frame Modification Data table index.

### 29.5.11.5.2 enum netc\_tb\_cmd\_t

Enumerator

*kNETC\_DeleteEntry* Delete operation.  
*kNETC\_UpdateEntry* Update operation.  
*kNETC\_QueryEntry* Query operation.  
*kNETC\_QueryAndDeleteEntry* Query operation followed by a delete operation.  
*kNETC\_QueryAndUpdateEntry* Query operation followed by a update operation.  
*kNETC\_AddEntry* Add operation.  
*kNETC\_AddOrUpdateEntry* If the entry exists, is update operation, if not exist, is the Add operation.  
*kNETC\_AddAndQueryEntry* Add operation followed by a query operation.  
*kNETC\_AddQueryAndUpdateEntry* Add operation followed by a query operation, Then, if the entry existed prior to the Add operation of this command, the Update operation will be performed.

### 29.5.11.5.3 enum netc\_tb\_access\_mode\_t

Enumerator

*kNETC\_EntryIDMatch* Entry ID Match.  
*kNETC\_ExactKeyMatch* Exact Match Key Element Match.  
*kNETC\_Search* Search with search criteria.  
*kNETC\_TernaryKeyMatch* Ternary Match Key Element Match.

### 29.5.11.5.4 enum netc\_cbd\_version\_t

Enumerator

*kNETC\_NtmpV1\_0* NTMP Version 1.0.  
*kNETC\_NtmpV2\_0* NTMP Version 2.0.

### 29.5.11.5.5 enum netc\_cmd\_error\_t

Enumerator

*kNETC\_FormatError* Format error : 1. Illegal class or command. 2. Invalid SF bit setting. 3. LENGTH is zero for long format. 4. LENGTH is too small for buffer size.  
*kNETC\_SizeError* Size error : 1. Invalid table index, out of range. 2. Table overflow, no additional entries available.  
*kNETC\_AccessError* Access violation error, the entity is not allowed to perform the task requested.  
*kNETC\_ClassError* Class specific error.

***kNETC\_IntegrityError*** Integrity error, the command did not execute due to a data integrity error (ECC on internal memory or AXI read/write error)

***kNETC\_InvTableID*** Invalid table ID.

***kNETC\_InvAccMethod*** Invalid Access method.

***kNETC\_TableIdxOutOfRange*** Table index out of range.

***kNETC\_DBNotEnough*** Request data buffer size or response data buffer size is not sufficient.

***kNETC\_InvCmd*** Invalid command.

***kNETC\_ReqDBError*** Request Data buffer error.

***kNETC\_MultiBitError*** Multi-bit ECC or parity error observed during command processing.

***kNETC\_HashEntryLimit*** Exceeded hash entry limit.

***kNETC\_HashChainLimit*** Exceeded maximum hash collision chain limit and the CAM if present is full.

***kNETC\_InvHWGenEntryID*** Invalid ENTRY\_ID for ENTRY\_ID generated by hardware.

***kNETC\_SrchResDBNotEnough*** Search command filled the response data buffer before completing the command.

***kNETC\_CmdIdxTableWithITM*** Command for index table before OSR[ITM\_STATE]=0.

***kNETC\_InvQueryAction*** Invalid Query action.

***kNETC\_InvTableAccPrivilege*** Invalid table access privilege.

***kNETC\_ReadSysBusErr*** System Bus Read Error.

***kNETC\_WriteSysBusErr*** System Bus Write Error.

***kNETC\_ClientErr*** Client encountered a fault.

***kNETC\_TGSCmdIssue*** Command issued when time gating function is disabled for the port.

***kNETC\_TGSUpdateExistList*** Update action attempted on an existing admin gate control list.  
(should delete admin gate control list first before creating a new admin list)

***kNETC\_TGSUpdateOverLength*** Update action attempted exceeds TGSTCAPR[MAX\_GCL\_LENGTH].

***kNETC\_TGSUpdateOverSize*** Update action attempted exceeds TGSTCAPR[NUM\_WORDS].

***kNETC\_TGSEntryNotEnough*** Insufficient resources to perform the requested operation (not enough free time gate list entries)

***kNETC\_TGSUpdateNSList*** Update action attempted with ADMIN\_CYCLE\_TIME, ADMIN\_TIME\_INTERVAL\_GE\_i or truncated ADMIN\_TIME\_INTERVAL\_GE\_n due ADMIN\_CYCLE\_TIME specified is not sufficient to transmit 64 byte of frame data + header overhead.

***kNETC\_TGSUpdateEarlierStartTime*** Update action attempted with ADMIN\_BASE\_TIME specified s more than one second in the past from tcs advance time.

***kNETC\_TGSUpdateOverflowCycle*** Update action attempted with ADMIN\_CYCLE\_TIME + ADMIN\_CYCLE\_TIME\_EXT is greater than  $2^{32}-1$ .

***kNETC\_TGSQueryBeforeListActive*** Query action issued when config change occurred. Retry query.

***kNETC\_TGSUpdateInvGateValue*** Update action attempted with ADMIN\_HR\_CB\_GE\_i set to an invalid value.

***kNETC\_RPSDUTypeOutOfRange*** SDU\_TYPE specified in entry CFGE\_DATA is out of range.

***kNETC\_IPFInvHR*** HR value not valid. Only checked if command issued from the Switch and FLTFA=0x2 or FLTFA=0x3

***kNETC\_IPFEntryNotFit*** Entry being added does not fit in table.

***kNETC\_IPFWithoutSTSE*** CFGE\_DATA update without STSE\_DATA update.

**kNETC\_IPFInvRPP** RPR set to a reserved value. Only checked if FLTA=0x2.

**kNETC\_IPFFLTATGTOutRange** FLTA\_TGT is outside valid range and not NULL. Only checked if FLTA>0x0

**kNETC\_IPFInvSwtFLTA** FLTA=0x3 when command issued from the Switch.

**kNETC\_IPFInvEnetcFLTA** FLTFA>0x1 when command issued from an ENETC PF.

**kNETC\_FDBReachPortLimit** Failed to add or update and entry because the Port BPCR[DYN\_LIMIT] has been reached.

**kNETC\_FDBReachSwtLimit** Failed to add entry because the Switch FDBHTMCR[DYN\_LIMIT] has been reached.

**kNETC\_FDBInvEPORT** EPORT value not valid. Only checked if (OETEID=0x1 OR CTD=0x1)

**kNETC\_FDBETEIDOutRange** ET\_EID is out of range and not NULL. Only checked if OETEID>0x0

**kNETC\_FDBParityErr** Parity error encountered when adding guaranteed entry.

**kNETC\_L2MCFInvEPORT** EPORT value not valid. Only checked if (OETEID=0x1 OR CTD=0x1)

**kNETC\_L2MCFETEIDOutRange** ET\_EID is not NULL or within the valid range. Only checked if OETEID>0x0.

**kNETC\_L2MCFInvKEYTYPE** KEY\_TYPE value not valid.

**kNETC\_VFBASEETEIDOutRange** BASE\_ET\_EID is out of range or MLO is not valid.

**kNETC\_ECQCQ2CGMAPOutRange** CQ2CG\_MAP value out-of-range in update command.

**kNETC\_ISIPortIDOutRange** Port ID specified in KEYE\_DATA is out of range.

**kNETC\_ISIInvISEID** IS\_EID in invalid.

**kNETC\_ISInvOpt** Option specified in one or more of the following fields is not valid – FA, CTD or ISQA, SDU\_TYPE.

**kNETC\_ISInvID** One or more of following : 1. Entry IDs are not in valid range or Entry ID is not Null. 2. Check valid ranges specified for these Entry IDs in Ingress Stream table entry – RP\_EID, SGI\_EID, ISQ\_EID, ET\_EID or EPORT. 3. ET\_EID is checked if (FA =010b .. 101b) & (OETEID!=0). 4. EPORT is checked if (FA = 010b .. 101b) & (OETEID= 0x1 OR CTD= 0x1). 5. HR is checked if FA = 001b, 100b, or 101b. HR specified cannot be 0x0

**kNETC\_ISInvFMEID** FM\_EID format or index is out of range : 1. FM\_EID format option type is invalid. 2. FM\_EID format is option 1 and the Index is out of range and not Null, or FM\_EID format is option 2 and VUDA or SQTA is out of range.

**kNETC\_ISFIInvISEID** IS\_EID in KEYE\_DATA is invalid.

**kNETC\_ISFIInvCFG** Any of the following in CFGE\_DATA is invalid : 1. One or more of following Entry IDs are not in valid range or Entry ID specified is not Null. Checks are performed for following Entry IDs CFGE DATA – RP\_EID, SGI\_EID, ISC\_EID. 2. SDU\_TYPE is invalid

**kNETC\_ETInvOpt** Command option specified is invalid or not supported. ESQA is not 00 or 10 (others are reserved), or ECA > 1 (reserved).

**kNETC\_ETInvFMEID** FM\_EID format or index is out of range. Check performed is as follows : 1. EFM\_EID format option type is invalid, or EFM\_EID format is option 1 and the Index is out of range and not Null, or EFM\_EID format is option 2 and VUDA or SQTA is out of range . 2. the Egress Counter Table index EC\_EID is out of range. 3. The Egress Sequence Actions Target Entry ID ESQA\_TGT\_EID is out of range

**kNETC\_ISGInvQSTAG** SQ\_TAG specified is not valid.

***kNETC\_SGISGCLEIDOutRange*** SGCL\_EID specified in out of range for Add or Update operation.

***kNETC\_SGIInvSDUTYPE*** SDU\_TYPE is specified is invalid for Add or Update operation.

***kNETC\_SGISGCLEIDNotAlloc*** Either the SGCL\_EID specified as admin gate control list in Add or Update operation has not been allocated or SGCL\_EID is not the first entry in gate control list or the reference count in SGCL entry is not 0.

***kNETC\_SGIInvSGCLEID*** SGCL\_EID specified for Update operation is in invalid.

***kNETC\_SGIInvADMINBASETIME*** ADMIN\_BASE\_TIME specified for Add or Update operation is more than  $2^{30}$ ns in the past.

***kNETC\_SGIInvCYCLETIME*** Cumulated time value of CYCLE\_TIME in Stream gate Control list plus CYCLE\_TIME\_EXT specified in Add or Update operation is  $>= 2^{30}$ ns or CYCLE\_TIME specified is 0.

***kNETC\_SGCLOverLength*** Number words required for the LIST\_LENGTH specified for the Add operation exceeds the number of words allocated for SGCL table.

***kNETC\_SGCLTimeIntervalZero*** TIME\_INTERVAL\_GE\_N specified in Add operation is 0. Note that upper 2 bits of TIME\_INTERVAL\_GE\_N are ignored, TIME\_INTERVAL\_GE\_N[29:0] must not be 0.

***kNETC\_SGCLTimeOverflow*** Cumulated time value of TIME\_INTERVAL\_GE\_N[29:0] for the gate list specified in Add operation is  $>= 2^{30}$ ns.

***kNETC\_FMInvEMEID*** FM\_EID format is invalid.

***kNETC\_FMOptOutRange*** Following fields specified are out of range - MAC\_HDR\_ACT, VLAN\_HDR\_ACT, SQT\_ACT, OUTER\_PCP\_DEI\_ACT, PLD\_ACT.

***kNETC\_FMFMDOutRange*** FMD\_EID,FMD\_BYTES specified is out of range. When FMD\_EID is not set to Null, valid range is FMD\_EID[15:0]\*24 + FMD\_BYTES  $\leq$  (FMDITCAPR[NUM\_WORDS]\*24).

***kNETC\_BPSBPEIDOutRange*** SBP\_EN is 1 and SBP\_EID value is out-of-range in update command.

### 29.5.11.5.6 enum netc\_fm\_vlan\_ud\_act\_t

Note

Misconfiguration error if replace or delete action is specified and if VLAN tag is not present in frame.

Enumerator

***kNETC\_NoUDVlanAction*** No Update/Delete VLAN action.

***kNETC\_ReplVlanPcpAndDei*** Replace outer VLAN's PCP/DEI based on the port's PPCPDEIMR. The tag's original VID and TPID are preserved

***kNETC\_DelVlan*** Delete outer VLAN Tag.

### 29.5.11.5.7 enum netc\_fm\_sqt\_act\_t

## Note

Must be set to 000b for Ingress frame modification, otherwise misconfiguration error..

## Enumerator

***kNETC\_NoSqtAction*** No SQT action.

***kNETC\_RemoveRTag*** Remove R-TAG/draft 2.0 R-TAG/HSR tag, If R-TAG/HSR tag not present, misconfiguration error.

**29.5.11.5.8 enum netc\_fm\_vlan\_ar\_act\_t**

## Note

For ingress frame modification with 00b or 01b, use the ingress port to select PCP and DEI from the Bridge port default VLAN register (BPDVR). For egress frame modification with 00b or 01b, use the internal QoS associated with the frame (IPV, DR) to access the QoS to PCP mapping profile (P-QOSMR[QVMP] , QOSVLANMPaR0/1/2/3) to set the new PCP value. Use internal DR associated with frame to access the DR to DEI mapping profile (PPCPDEIMR[DRnDEI]) to set the new DEI value.

## Enumerator

***kNETC\_AddCVlanPcpAndDei*** Add outer VLAN with VID and PCP/DEI updated as described above. TPID=0x8100

***kNETC\_AddSVlanPcpAndDei*** Add outer VLAN with VID and PCP/DEI updated as described above. TPID=0x88A8

***kNETC\_ReplVidOnly*** Replace VLAN with VID. The tag's original PCP, DEI and TPID are preserved

***kNETC\_ReplVidPcpAndDei*** Replace VLAN with VID and PCP/DEI updated by port's PPCPDEIMR. The tag's original TPID is preserved

**29.5.11.5.9 enum netc\_tb\_eteid\_access\_mode\_t**

## Note

The FDB/L2 IPv4 Multicast filter table has precedence over any assignment made via the Ingress Stream table. For Multi port mode, the index to access the Egress Treatment table is computed by adding an offset to the base index of the Egress Treatment group. That offset is derived from the applicability bitmap as follows: starting from the lowest significant bit of the bitmap, the first encountered bit set to 1, corresponds to offset 0, and so on. This continues till the destination port location in the bitmap is reached

## Enumerator

***kNETC\_NoETAccess*** No Egress Treatment table access.

***kNETC\_SinglePortETAccess*** Only frame sent to a special port (define in ePort) can access a single Egress Treatment table entry, the applicability bitmap specified by FDB/L2MCF/IS ePort field.

***kNETC\_MulitPortPackedETAccess*** Only frames sent to a special set of ports (ports set to 1 in e-PortBitmap) can access the Egress Treatment table, the applicability bitmap = IS ePortBitmap field or FDB/L2MCF portBitmap field.

***kNETC\_MulitPortAbsETAccess*** Frames sent to all of ports can access the Egress Treatment table, means the applicability bitmap is set with 1 for all ports.

#### 29.5.11.5.10 enum netc\_tb\_ipf\_update\_action\_t

Enumerator

***kNETC\_IPFCfgEUpdate*** Configuration Element Update.

***kNETC\_IPFStsEUpdate*** Statistics Element Update.

#### 29.5.11.5.11 enum netc\_tb\_ipf\_attr\_mask\_t

Enumerator

***kNETC\_IPFSwtPortMasMask*** Switch port masquerading Mask.

***kNETC\_IPFEthernetMask*** Ethernet type Mask.

***kNETC\_IPFOuterVlanMask*** Outer VLAN Mask.

***kNETC\_IPFInnerVlanMask*** Inner VLAN Mask.

***kNETC\_IPFSeqTagMask*** Sequence Tag Code Mask.

***kNETC\_IPFIpHeaderMask*** IP Header Mask.

***kNETC\_IPFIpVersionMask*** IP Version Mask.

***kNETC\_IPFIpExtMask*** IPv4 option / IPv6 extension Mask.

***kNETC\_IPFL4HeaderMask*** L4 Code Mask.

***kNETC\_IPFWakeOnLanMask*** Wake-on-LAN Magic Packet Mask.

#### 29.5.11.5.12 enum netc\_tb\_ipf\_seq\_tag\_t

Enumerator

***kNETC\_IPFNoRtag*** R-TAG/HSR tag is not present.

***kNETC\_IPFDraftRtag*** 802.1CB draft 2.0 R-TAG is present

***kNETC\_IPFRtag*** 802.1CB R-TAG is present

***kNETC\_IPFHsrTag*** HSR Tag is present.

**29.5.11.5.13 enum netc\_tb\_ipf\_l4\_header\_t**

Enumerator

***kNETC\_IPFOtherL4*** The L4 Header is considered as other L4 if it is not one of the following L4 Headers.

***kNETC\_IPFTcp*** TCP header is present.

***kNETC\_IPFUsip*** UDP header is present.

***kNETC\_IPFSctp*** SCTP header is present.

**29.5.11.5.14 enum netc\_tb\_ipf\_forward\_action\_t**

Enumerator

***kNETC\_IPFForwardDiscard*** Frame be discard.

***kNETC\_IPFForwardPermit*** Frame be permit.

***kNETC\_IPFRedirectToMgmtPort*** Redirect frame to switch management port without any frame modification, Switch only.

***kNETC\_IPFCopyToMgmtPort*** Copy frame to switch management port without any frame modification, Switch only.

**29.5.11.5.15 enum netc\_tb\_ipf\_filter\_action\_t**

Enumerator

***kNETC\_IPFNoAction*** No action.

***kNETC\_IPFWithRatePolicer*** Rate action with the Rate Policer Entry ID (RP\_EID) set to the value configured in the fltaTgt field.

***kNETC\_IPFWithIngressStream*** Ingress stream identification action where the Ingress Stream Entry ID (IS\_EID) is set to the value configured in the fltaTgt field.

***kNETC\_IPFWithL2Filtering*** Setting a pre L2 filtering SI bitmap (set to the value configured in the fltaTgt) that will be used by the L2 filtering function to determine the final SI bitmap, ENETC only.

**29.5.11.5.16 enum netc\_tb\_isi\_key\_type**

Enumerator

***kNETC\_KCRule0*** Use key construction rule 0 (ISIDKC0CR0)

***kNETC\_KCRule1*** Use key construction rule 1 (ISIDKC1CR0)

***kNETC\_KCRule2*** Use key construction rule 2 (ISIDKC2CR0). Only for SWITCH

***kNETC\_KCRule3*** Use key construction rule 3 (ISIDKC3CR0). Only for SWITCH

**29.5.11.5.17 enum netc\_tb\_is\_isq\_action\_t**

Enumerator

*kNETC\_ISNotPerformFRER* Not perform Frame FRER sequence generation function.*kNETC\_ISPerformFRER* Perform Frame FRER sequence generation function.**29.5.11.5.18 enum netc\_tb\_is\_forward\_action\_t**

Enumerator

*kNETC\_ISDiscard* Frame be discard.*kNETC\_ISRedirectToMgmtPort* Frame be Re-direct frame to switch management port, Switch only.*kNETC\_ISAllow* Frame is allow without setting the pre L2 filtering SI bitmap, ENETC only.*kNETC\_ISAllowWithSIMap* Frame is allow with setting the pre L2 filtering SI bitmap to the value configured in the SI\_MAP field, ENETC only.*kNETC\_ISStreamForward* Frame is forwarded to the port(s) specified in the EGRESS\_PORT\_BITMAP field, Switch only.*kNETC\_ISBridgeForward* Frame is do 802.1Q Bridge forwarding (VLAN processing and L2 forwarding), Switch only.*kNETC\_ISCopyToMgmtPortAndStream* Copy frame to switch management port with specified HR and stream forwarding, Switch only.*kNETC\_ISCopyToMgmtPortAndBridge* Copy frame to switch management port with specified HR and Bridge forwarding, Switch only.**29.5.11.5.19 enum netc\_tb\_is\_ctd\_mode\_t**

Enumerator

*kNETC\_ISNoCTD* Do not override cut-through state.*kNETC\_ISSinglePortCTD* Disable cut-through for the outgoing port specified in the cfge ePort field.*kNETC\_ISAllPortCTD* Disable cut-through for all ports specified in cfge portBitmap field.**29.5.11.5.20 enum netc\_tb\_rp\_update\_action\_t**

Enumerator

*kNETC\_RPCfgEUpdate* Configuration Element Update.*kNETC\_RPFcEUpdate* Functional Enable Element Update.*kNETC\_RPPsEUpdate* Policer State Element Update Element Update.*kNETC\_RPStsEUpdate* Statistics Element Update.

**29.5.11.5.21 enum netc\_tb\_sgi\_update\_action\_t**

Enumerator

- kNETC\_SGIAcfEUpdate*** Admin Configuration Element.
- kNETC\_SGICfgEUpdate*** Configuration Element Update.
- kNETC\_SGISgisEUpdate*** Stream Gate Instance State Element Update.

**29.5.11.5.22 enum netc\_tb\_sgi\_state\_t**

Enumerator

- kNETC\_GSNotOper*** Gate instance is not operational or Gate instance and lists are not valid.
- kNETC\_GSUseDefaultParam*** Gate instance is operational but no stream gate control list specified, use default Gate Instance parameters.
- kNETC\_GSUseDefUntilAdminAct*** Use default Gate Instance parameters until administrative stream gate control list takes effect.
- kNETC\_GSUseOperUntilAdminAct*** Use Operational stream gate control list until new administrative stream gate control list takes effect.
- kNETC\_GSUseOperList*** Operational stream gate control list is in effect.

**29.5.11.5.23 enum netc\_tb\_fm\_layer2\_act\_t**

Note

This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

Enumerator

- kNETC\_UseL2HeaderAct*** L2 actions are specified in L2 header action fields macHdrAct, vlanHdrAct and sqtAct.
- kNETC\_UseSpecPayload*** The entire L2 PDU is replaced with fmdBytes of data specified in fmd-EID, not applicable for ingress frame modifications.

**29.5.11.5.24 enum netc\_tb\_fm\_mac\_header\_act\_t**

Note

Ingress frame modifications only support kNETC\_NoAction or kNETC\_ReplDmac.

Enumerator

- kNETC\_NoMacAction*** No Mac header action.

***kNETC\_ReplSmac*** Replace SMAC with the contents of the port's PMAR0/1 register, The port is specified by smacPort field.

***kNETC\_ReplSmacAndDmacAct1*** Replace SMAC and DMAC, The content of SMAC is the same as kNETC\_ReplaceSMAC, the DMAC is specified by dmac[6] field.

***kNETC\_ReplSmacAndDmacAct2*** Replace SMAC and DMAC, The content of SMAC is the same as kNETC\_ReplaceSMAC, the DMAC is specified by frame's SMAC.

***kNETC\_ReplDmac*** Replace DMAC with specified dmac[6] field value.

***kNETC\_SwapDmacAndSmac*** Swap DMAC and SMAC.

#### 29.5.11.5.25 enum netc\_tb\_fm\_vlan\_header\_act\_t

Note

For use Delete or Replace action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Enumerator

***kNETC\_NoVlanAction*** No VLAN header action.

***kNETC\_DelOuterVlan*** Delete outer VLAN header.

***kNETC\_AddOuterVlan*** Add outer VLAN header (new VLAN data will be inserted in the outer position), the VID, PCP, DEI and TPID values are specified by outerVidAct, outerPcpAct, outerDeiAct and outerTpidAct field.

***kNETC\_ReplOuterVlan*** Replace outer VLAN header, the VID, PCP, DEI and TPID values are specified by outerVidAct, outerPcpAct, outerDeiAct and outerTpidAct field.

#### 29.5.11.5.26 enum netc\_tb\_fm\_outer\_vid\_act\_t

Note

For use kNETC\_UseFrameVID action, if no outer VLAN header is present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

Enumerator

***kNETC\_UseFrameVid*** Use the VID from the valid outer VLAN header of the received frame.

***kNETC\_UseSpecVid*** Use the VID specified in the outerVlanID field.

#### 29.5.11.5.27 enum netc\_tb\_fm\_outer\_tpid\_act\_t

## Note

For use kNETC\_UseFrameTpid action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

## Enumerator

***kNETC\_UseFrameTpid*** Use TPID from outer VLAN header.

***kNETC\_UseStdCVlan*** Set TPID to Standard C-VLAN 0x8100.

***kNETC\_UseStdSVlan*** Set TPID to Standard S-VLAN 0x88A8.

***kNETC\_UseCustomCVlan*** Set TPID to Custom C-VLAN as defined by CVLANR1[ETYPE].

***kNETC\_UseCustomSVlan*** Set TPID to Custom S-VLAN as defined by CVLANR2[ETYPE].

**29.5.11.5.28 enum netc\_tb\_fm\_outer\_pcp\_act\_t**

## Note

For use kNETC\_UseFramePcp/kNETC\_UseFramePcpMap action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's P-FMCR register.

## Enumerator

***kNETC\_UseFramePcp*** Use PCP from frame outer VLAN header.

***kNETC\_UseSpecPcp*** Use the PCP specified in the outerVlanPcp field.

***kNETC\_UseFramePcpMap*** The PCP is mapping from frame outer VLAN PCP (do mapping according to the PCP to PCP mapping profile which specified in PPCPDEIMR[IPCPMP/EP-CPMP])

***kNETC\_UseQosMap*** The PCP is mapping from internal QoS (IPV, DR) (do mapping according to the QOS to PCP mapping profile which specified in QOSVLANMPaR0/1/2/3), not applicable for ingress frame modifications.

**29.5.11.5.29 enum netc\_tb\_fm\_outer\_dei\_act\_t**

## Note

For use kNETC\_UseFrameDei action, If outer VLAN header not present, then a misconfiguration event will be generated and handled according to the port's PFMCR register.

## Enumerator

***kNETC\_UseFrameDei*** Use DEI from frame outer VLAN header.

***kNETC\_UseSpecDei*** Use the DEI specified in the outerVlanDei field.

***kNETC\_UseDrMap*** The DEI is mapping from internal DR (do mapping according to the DR to DEI mapping profile which specified in PPCPDEIMR[DRnDEI]), not applicable for ingress frame modifications.

**29.5.11.5.30 enum netc\_tb\_fm\_payload\_act\_t**

Note

This field must be set to 0 for traffic destined to a pseudo link. This field must be set to 0 for any device with ASIL-B safety requirements.

Enumerator

*kNETC\_NoAction* No Action.

*kNETC\_ReplAllEthPld* Remove entire Ethernet payload and insert with fmdBytes of data specified in fmdEID.

*kNETC\_ReplPldWithOffset* Replace fmdBytes of raw data in the Ethernet payload starting at pld-Offset, data specified in fmdEID.

**29.5.11.5.31 enum netc\_tb\_fdb\_update\_action\_t**

Enumerator

*kNETC\_FDBCfgEUpdate* Configuration Element Update.

*kNETC\_FDBActEUpdate* Activity Element Update.

**29.5.11.5.32 enum netc\_tb\_fdb\_ctd\_mode\_t**

Enumerator

*kNETC\_FDBNoCTD* Do not override cut-through state.

*kNETC\_FDBSinglePortCTD* Disable cut-through for the outgoing port specified in the cfge ePort field.

*kNETC\_FDBAllPortCTD* Disable cut-through for all ports specified in cfge portBitmap field.

**29.5.11.5.33 enum netc\_tb\_fdb\_sc\_keye\_mc\_t**

Enumerator

*kNETC\_FDBKeyeMacthAny* Match any Key Element Criteria.

*kNETC\_FDBKeyeMacthFID* Match Key Element FID.

*kNETC\_FDBKeyeMacthMacMulticast* Match Key Element MAC Multicast bit (MAC\_ADDR most significant byte's least significant bit)

*kNETC\_FDBKeyeMacthBoth* Match both FID field and MAC Multicast bit.

**29.5.11.5.34 enum netc\_tb\_fdb\_sc\_cfge\_mc\_t**

Enumerator

*kNETC\_FDBCfgeMacthAny* Match any Configuration Element Criteria.*kNETC\_FDBCfgeMacthDynamic* Match Configuration Element dynamic field.*kNETC\_FDBCfgeMacthPortBitmap* Match Configuration Element portBitmap field.*kNETC\_FDBCfgeMacthBoth* Match both dynamic field and portBitmap.**29.5.11.5.35 enum netc\_tb\_fdb\_sc\_acte\_mc\_t**

Enumerator

*kNETC\_FDBActeMacthAny* Match any Activity Element Criteria.*kNETC\_FDBActeMatchExact* Exact match with Activity Element.**29.5.11.5.36 enum netc\_tb\_l2mcf\_key\_type\_t**

Enumerator

*kNETC\_IPv4ASMKey* Key consists of a filtering ID (FID) and destination multicast IPv4 address.*kNETC\_IPv4SSMKey* Key consists of a filtering ID (FID), IPv4 source address and multicast IPv4 destination address.**29.5.11.5.37 enum etc\_tb\_l2mcf\_sc\_keye\_mc\_t**

Enumerator

*kNETC\_L2MCFKeyeMacthAny* Match any Key Element Criteria.*kNETC\_L2MCFKeyeMacthFID* Match Key Element FID.**29.5.11.5.38 enum etc\_tb\_l2mcf\_sc\_cfge\_mc\_t**

Enumerator

*kNETC\_L2MCFCfgemacthAny* Match any Configuration Element Criteria.*kNETC\_L2MCFCfgemacthDynamic* Match Configuration Element dynamic field.*kNETC\_L2MCFCfgemacthPortBitmap* Match Configuration Element portBitmap field.*kNETC\_L2MCFCfgemacthBoth* Match both dynamic field and portBitmap.

**29.5.11.5.39 enum etc\_tb\_l2mcf\_sc\_acte\_mc\_t**

Enumerator

*kNETC\_L2MCFActeMatchAny* Match any Activity Element Criteria.  
*kNETC\_L2MCFActeMatchExact* Exact match with Activity Element.

**29.5.11.5.40 enum netc\_tb\_iseqg\_sqtag\_t**

Enumerator

*kNETC\_SqDraftRTag* 802.1CB draft 2.0 R-TAG.  
*kNETC\_SqRTag* 802.1CB R-TAG.  
*kNETC\_SqHsrTag* HSR Tag.

**29.5.11.5.41 enum netc\_tb\_eseqr\_sqtag\_t**

Enumerator

*kNETC\_AcceptAnyTag* Accept any incoming tag type (802.1CB draft 2.0 R-TAG, 802.1CB R-TAG or HSR Tag)  
*kNETC\_AcceptDraftRTag* 802.1CB draft 2.0 R-TAG.  
*kNETC\_AcceptRTag* 802.1CB R-TAG.  
*kNETC\_AcceptHsrTag* HSR Tag.

**29.5.11.5.42 enum netc\_tb\_tgs\_entry\_id\_t**

Enumerator

*kNETC\_TGSSwtPort0* Switch PORT 0 entry ID.  
*kNETC\_TGSSwtPort1* Switch PORT 1 entry ID.  
*kNETC\_TGSSwtPort2* Switch PORT 2 entry ID.  
*kNETC\_TGSSwtPort3* Switch PORT 3 entry ID.  
*kNETC\_TGSSwtPort4* Switch PORT 4 entry ID.  
*kNETC\_TGSEnetc0Port* ENETC 0 port entry ID.  
*kNETC\_TGSEnetc1Port* ENETC 1 port entry ID.

**29.5.11.5.43 enum netc\_tb\_tgs\_gate\_type\_t**

Enumerator

*kNETC\_SetGateStates* HoldRequest is unchanged.  
*kNETC\_SetAndHoldMac* HoldRequest is set to value hold, only active when enable preemption.  
*kNETC\_SetAndReleaseMac* HoldRequest is set to value release, only active when enable preemption.

#### 29.5.11.5.44 enum netc\_tb\_et\_efm\_mode\_t

Enumerator

*kNETC\_NormalMode* Egress Frame Modification entry use normal/Default mode.

*kNETC\_L2Act1* Egress Frame Modification entry l2Act = kNETC\_UseSpecPayload.

*kNETC\_PldAct1* Egress Frame Modification entry pldAct = kNETC\_ReplAllEthPld.

#### 29.5.11.5.45 enum netc\_tb\_et\_esq\_act\_t

Enumerator

*kNETC\_NoEsqAction* No Egress Sequence Action required.

*kNETC\_HasEsqAction* Has Egress Sequence Recovery action.

#### 29.5.11.5.46 enum netc\_tb\_et\_ec\_act\_t

Enumerator

*kNETC\_NoEcCounter* Do not increment egress frame counter.

*kNETC\_HasEcCounter* Increment egress frame counter.

#### 29.5.11.5.47 enum netc\_tb\_etmcq\_update\_action\_t

Enumerator

*kNETC\_CQCfgEUpdate* Configuration Element Update.

*kNETC\_CQStsEUpdate* Statistics Element Update, all counters (except FRM\_CNT) within the Statistics Element are reset.

#### 29.5.11.5.48 enum netc\_tb\_etmcs\_entry\_id\_t

Enumerator

*kNETC\_CSSwtPort0* CS Switch PORT 0 entry ID.

*kNETC\_CSSwtPort1* CS Switch PORT 1 entry ID.

*kNETC\_CSSwtPort2* CS Switch PORT 2 entry ID.

*kNETC\_CSSwtPort3* CS Switch PORT 3 entry ID.

*kNETC\_CSSwtPort4* CS Switch PORT 4 entry ID.

### 29.5.11.5.49 enum netc\_tb\_etmes\_ca\_assg\_t

Enumerator

*kNETC\_CQ7AssignToSchedIn15* CQ 7 assignment to scheduler input 15, means all CQ use strict priority.  
*kNETC\_CQ7AssignToSchedIn14* CQ 7 assignment to scheduler input 14.  
*kNETC\_CQ7AssignToSchedIn13* CQ 7 assignment to scheduler input 13.  
*kNETC\_CQ7AssignToSchedIn12* CQ 7 assignment to scheduler input 12.  
*kNETC\_CQ7AssignToSchedIn11* CQ 7 assignment to scheduler input 11.  
*kNETC\_CQ7AssignToSchedIn10* CQ 7 assignment to scheduler input 10.  
*kNETC\_CQ7AssignToSchedIn9* CQ 7 assignment to scheduler input 9.  
*kNETC\_CQ7AssignToSchedIn8* CQ 7 assignment to scheduler input 8.  
*kNETC\_CQ7AssignToSchedIn7* CQ 7 assignment to scheduler input 7, means all CQ use weighted fair.

### 29.5.11.5.50 enum netc\_tb\_bp\_fc\_cfg\_t

Enumerator

*kNETC\_FlowCtrlDisable* Flow Control disabled.  
*kNETC\_FlowCtrlWithBP* Flow Control enabled using only buffer pool FC state.  
*kNETC\_FlowCtrlWithSBP* Flow Control enabled using only shared buffer pool FC state.  
*kNETC\_FlowCtrlWithBPAAndSBP* Flow Control enabled using both buffer pool and shared buffer pool FC state, only both 1 trigger the Flow Control ON.

## 29.5.11.6 Function Documentation

### 29.5.11.6.1 status\_t NETC\_CmdBDRInit ( *netc\_cbdr\_hw\_t \* base*, *const netc\_cmd\_bdr\_config\_t \* config* )

Parameters

|               |  |
|---------------|--|
| <i>base</i>   |  |
| <i>config</i> |  |

Returns

*kStatus\_Success*  
*kStatus\_Fail*

### 29.5.11.6.2 status\_t NETC\_CmdBDRDeinit ( *netc\_cbdr\_hw\_t \* base* )

Parameters

|             |
|-------------|
| <i>base</i> |
|-------------|

Returns

kStatus\_Success

#### 29.5.11.6.3 status\_t NETC\_CmdBDSendCommand ( netc\_cbdr\_hw\_t \* *base*, netc\_cmd\_bdr\_t \* *cbdr*, netc\_cmd\_bd\_t \* *cbd*, netc\_cbd\_version\_t *version* )

Parameters

|                |  |
|----------------|--|
| <i>base</i>    |  |
| <i>cbdr</i>    |  |
| <i>cbd</i>     |  |
| <i>version</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.4 status\_t NETC\_AddIPTableEntry ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_ipf\_config\_t \* *config*, uint32\_t \* *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>config</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.5 status\_t NETC\_UpdateIPTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_ipf\_cfg\_t \* *cfg* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>cfg</i>     |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.6 status\_t NETC\_QueryIPFTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_ipf\_config\_t \* *config* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>config</i>  |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.7 status\_t NETC\_DellIPFTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.8 status\_t NETC\_ResetIPFMatchCounter ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.9 status\_t NETC\_GetIPFMatchedCount ( *netc\_cbdr\_handle\_t \* handle, uint32\_t entryID, uint64\_t \* count* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>count</i>   |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.10 status\_t NETC\_AddISITableEntry ( *netc\_cbdr\_handle\_t \* handle, netc\_tb\_isi\_config\_t \* config, uint32\_t \* entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>config</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.11 status\_t NETC\_DelISITableEntry ( *netc\_cbdr\_handle\_t \* handle, uint32\_t entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.12 status\_t NETC\_QueryISITableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_isi\_config\_t \* *config* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>config</i>  |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.13 status\_t NETC\_QueryISITableEntryWithKey ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_isi\_keye\_t \* *keye*, netc\_tb\_isi\_rsp\_data\_t \* *rsp* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>keye</i>   |  |
| <i>rsp</i>    |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.14 status\_t NETC\_AddOrUpdateISTableEntry ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_is\_config\_t \* *config*, bool *isAdd* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>config</i> |  |
| <i>isAdd</i>  |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.15 status\_t NETC\_QueryISTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_is\_config\_t \* *config* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>config</i>  |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.16 status\_t NETC\_DelISTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.17 status\_t NETC\_AddISFTableEntry ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_isf\_config\_t \* *config*, uint32\_t \* *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>config</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.18 status\_t NETC\_UpdateISFTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_isf\_cfge\_t \* *cfg* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>cfg</i>     |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.19 status\_t NETC\_DelISFTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.20 status\_t NETC\_QueryISFTableEntry ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_isf\_keye\_t \* *keye*, netc\_tb\_isf\_rsp\_data\_t \* *rsp* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>keye</i>   |  |
| <i>rsp</i>    |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.21 status\_t NETC\_AddISCTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

#### 29.5.11.6.22 status\_t NETC\_GetISCStatistic ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_isc\_stse\_t \* *statistic* )

Parameters

|                  |  |
|------------------|--|
| <i>handle</i>    |  |
| <i>entryID</i>   |  |
| <i>statistic</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.23 status\_t NETC\_ResetISCStatistic ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

**29.5.11.6.24 status\_t NETC\_AddOrUpdateSGITableEntry ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_sgi\_config\_t \* *config*, bool *isAdd* )**

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>config</i> |  |
| <i>isAdd</i>  |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

**29.5.11.6.25 status\_t NETC\_DelSGITableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )**

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

**29.5.11.6.26 status\_t NETC\_GetSGIState ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_sgi\_sgise\_t \* *statis* )**

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>statis</i>  |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.27 status\_t NETC\_QuerySGITableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_sgi\_rsp\_data\_t \* *rsp* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>rsp</i>     |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.28 status\_t NETC\_AddSGCLTableEntry ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_sgcl\_gcl\_t \* *config* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>config</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.29 status\_t NETC\_DelSGCLTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.30 status\_t NETC\_GetSGCLGateList ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_sgcl\_gcl\_t \* *gcl*, uint32\_t *length* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>gcl</i>    |  |
| <i>length</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.31 status\_t NETC\_GetSGCLState ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_sgcl\_sgclse\_t \* *state* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>state</i>   |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.32 status\_t NETC\_QueryRPTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_rp\_rsp\_data\_t \* *rsp* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>rsp</i>     |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.33 status\_t NETC\_AddOrUpdateRPTableEntry ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_rp\_config\_t \* *config*, netc\_tb\_cmd\_t *cmd* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>config</i> |  |
| <i>cmd</i>    |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.34 status\_t NETC\_DelRPTableEntry ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.35 status\_t NETC\_GetRPStatistic ( netc\_cbdr\_handle\_t \* *handle*, uint32\_t *entryID*, netc\_tb\_rp\_stse\_t \* *statis* )

Parameters

|                |  |
|----------------|--|
| <i>handle</i>  |  |
| <i>entryID</i> |  |
| <i>statis</i>  |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.36 status\_t NETC\_ConfigTGSAdminList ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_tgs\_gcl\_t \* *config* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>config</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

#### 29.5.11.6.37 status\_t NETC\_GetTGSOperationList ( netc\_cbdr\_handle\_t \* *handle*, netc\_tb\_tgs\_gcl\_t \* *gcl*, uint32\_t *length* )

Parameters

|               |  |
|---------------|--|
| <i>handle</i> |  |
| <i>gcl</i>    |  |
| <i>length</i> |  |

Returns

status\_t

See [netc\\_cmd\\_error\\_t](#)

## 29.6 Abbreviation in NETC driver

Abbreviation list in NETC driver

- UC: unicast
- BC: broadcast
- MC: multicast
- IPF: ingress port filter
- VlanC: Vlan classification
- QosC: Qos Classification
- FRER: Frame Replication and Elimination for Reliability
- PSFP: Per-stream Filtering and Policing
- FM: Frame Modification
- SGI: Stream Gate Instance
- SGC: Stream Gate Control List
- ISI: Ingress Stream Identification
- ISC: Ingress stream count
- IS: Ingress Stream Table
- VF: Vlan Filter
- MF: Mac Filter
- EM: Extract Match

# Chapter 30

## PDM: Microphone Interface

### 30.1 Overview

#### Modules

- PDM Driver

## 30.2 PDM Driver

### 30.2.1 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.

### 30.2.2 Typical use case

#### 30.2.2.1 PDM receive using an interrupt method

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

#### 30.2.2.2 PDM receive using a SDMA method

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

#### 30.2.2.3 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

### 30.2.2.4 PDM receive using a transactional method

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

## Data Structures

- struct [pdm\\_channel\\_config\\_t](#)  
*PDM channel configurations.* [More...](#)
- struct [pdm\\_config\\_t](#)  
*PDM user configuration structure.* [More...](#)
- struct [pdm\\_hwvad\\_config\\_t](#)  
*PDM voice activity detector user configuration structure.* [More...](#)
- struct [pdm\\_hwvad\\_noise\\_filter\\_t](#)  
*PDM voice activity detector noise filter user configuration structure.* [More...](#)
- struct [pdm\\_hwvad\\_zero\\_cross\\_detector\\_t](#)  
*PDM voice activity detector zero cross detector configuration structure.* [More...](#)
- struct [pdm\\_transfer\\_t](#)  
*PDM SDMA transfer structure.* [More...](#)
- struct [pdm\\_hwvad\\_notification\\_t](#)  
*PDM HWVAD notification structure.* [More...](#)
- struct [pdm\\_handle\\_t](#)  
*PDM handle structure.* [More...](#)

## Macros

- #define [PDM\\_XFER\\_QUEUE\\_SIZE](#) (4U)  
*PDM XFER QUEUE SIZE.*

## Typedefs

- typedef void(\* [pdm\\_transfer\\_callback\\_t](#) )(PDM\_Type \*base, pdm\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*PDM transfer callback prototype.*
- typedef void(\* [pdm\\_hwvad\\_callback\\_t](#) )(status\_t status, void \*userData)  
*PDM HWVAD callback prototype.*

## Enumerations

- enum {
   
kStatus\_PDM\_Busy = MAKE\_STATUS(kStatusGroup\_PDM, 0),
   
kStatus\_PDM\_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_range_status` {

```

kPDM_RangeStatusUnderFlowCh0 = PDM_RANGE_STAT_RANGEUNF0_MASK,
kPDM_RangeStatusUnderFlowCh1 = PDM_RANGE_STAT_RANGEUNF1_MASK,
kPDM_RangeStatusUnderFlowCh2 = PDM_RANGE_STAT_RANGEUNF2_MASK,
kPDM_RangeStatusUnderFlowCh3 = PDM_RANGE_STAT_RANGEUNF3_MASK,
kPDM_RangeStatusUnderFlowCh4 = PDM_RANGE_STAT_RANGEUNF4_MASK,
kPDM_RangeStatusUnderFlowCh5 = PDM_RANGE_STAT_RANGEUNF5_MASK,
kPDM_RangeStatusUnderFlowCh6 = PDM_RANGE_STAT_RANGEUNF6_MASK,
kPDM_RangeStatusUnderFlowCh7 = PDM_RANGE_STAT_RANGEUNF7_MASK,
kPDM_RangeStatusOverFlowCh0 = PDM_RANGE_STAT_RANGEOVF0_MASK,
kPDM_RangeStatusOverFlowCh1 = PDM_RANGE_STAT_RANGEOVF1_MASK,
kPDM_RangeStatusOverFlowCh2 = PDM_RANGE_STAT_RANGEOVF2_MASK,
kPDM_RangeStatusOverFlowCh3 = PDM_RANGE_STAT_RANGEOVF3_MASK,
kPDM_RangeStatusOverFlowCh4 = PDM_RANGE_STAT_RANGEOVF4_MASK,
kPDM_RangeStatusOverFlowCh5 = PDM_RANGE_STAT_RANGEOVF5_MASK,
kPDM_RangeStatusOverFlowCh6 = PDM_RANGE_STAT_RANGEOVF6_MASK,
kPDM_RangeStatusOverFlowCh7 = PDM_RANGE_STAT_RANGEOVF7_MASK }

```

*The PDM output status.*

- enum `pdm_dc_remover_t` {

```

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

```

*PDM DC remover configurations.*

- enum `pdm_df_quality_mode_t` {

```
kPDM_QualityModeMedium = 0U,
kPDM_QualityModeHigh = 1U,
kPDM_QualityModeLow = 7U,
kPDM_QualityModeVeryLow0 = 6U,
kPDM_QualityModeVeryLow1 = 5U,
kPDM_QualityModeVeryLow2 = 4U }
```

*PDM decimation filter quality mode.*

- enum `_pdm_qulaity_mode_k_factor` {
 

```
kPDM_QualityModeHighKFactor = 1U,
kPDM_QualityModeMediumKFactor = 2U,
kPDM_QualityModeLowKFactor = 4U,
kPDM_QualityModeVeryLow2KFactor = 8U }
```

*PDM quality mode K factor.*

- enum `pdm_df_output_gain_t` {
 

```
kPDM_DfOutputGain0 = 0U,
kPDM_DfOutputGain1 = 1U,
kPDM_DfOutputGain2 = 2U,
kPDM_DfOutputGain3 = 3U,
kPDM_DfOutputGain4 = 4U,
kPDM_DfOutputGain5 = 5U,
kPDM_DfOutputGain6 = 6U,
kPDM_DfOutputGain7 = 7U,
kPDM_DfOutputGain8 = 8U,
kPDM_DfOutputGain9 = 9U,
kPDM_DfOutputGain10 = 0xAU,
kPDM_DfOutputGain11 = 0xBU,
kPDM_DfOutputGain12 = 0xCU,
kPDM_DfOutputGain13 = 0xDU,
kPDM_DfOutputGain14 = 0xEU,
kPDM_DfOutputGain15 = 0xFU }
```

*PDM decimation filter output gain.*

- enum `_pdm_data_width` {
 

```
kPDM_DataWidth24 = 3U,
kPDM_DataWidth32 = 4U }
```
- PDM data width.*
- enum `_pdm_hwvad_interrupt_enable` {
 

```
kPDM_HwvadErrorInterruptEnable = PDM_VAD0_CTRL_1_VADERIE_MASK,
kPDM_HwvadInterruptEnable = PDM_VAD0_CTRL_1_VADIE_MASK }
```

*PDM voice activity detector interrupt type.*

- enum `_pdm_hwvad_int_status` {
 

```
kPDM_HwvadStatusInputSaturation = PDM_VAD0_STAT_VADINSATF_MASK,
kPDM_HwvadStatusVoiceDetectFlag = PDM_VAD0_STAT_VADIF_MASK }
```

*The PDM hwvad interrupt status flag.*

- enum `pdm_hwvad_hpf_config_t` {

- kPDM\_HwvadHpfBypassed = 0x0U,  
 kPDM\_HwvadHpfCutOffFreq1750Hz = 0x1U,  
 kPDM\_HwvadHpfCutOffFreq215Hz = 0x2U,  
 kPDM\_HwvadHpfCutOffFreq102Hz = 0x3U }
- High pass filter configure cut-off frequency.*
- enum **pdm\_hwvad\_filter\_status\_t** {  
 kPDM\_HwvadInternalFilterNormalOperation = 0U,  
 kPDM\_HwvadInternalFilterInitial = PDM\_VAD0\_CTRL\_1\_VADST10\_MASK }
- HWVAD internal filter status.*
- enum **pdm\_hwvad\_zcd\_result\_t** {  
 kPDM\_HwvadResultOREnergyBasedDetection,  
 kPDM\_HwvadResultANDEnergyBasedDetection }
- PDM voice activity detector zero cross detector result.*

## Driver version

- #define **FSL\_PDM\_DRIVER\_VERSION** (MAKE\_VERSION(2, 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_GetRangeStatus (PDM_Type *base)`  
*Gets the PDM Range 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_ClearRangeStatus (PDM_Type *base, uint32_t mask)`  
*Clears the PDM range status.*

## Interrupts

- `void PDM_EnableInterrupts (PDM_Type *base, uint32_t mask)`  
*Enables the PDM interrupt requests.*
- `static void PDM_DisableInterrupts (PDM_Type *base, uint32_t mask)`  
*Disables the PDM interrupt requests.*

## DMA Control

- `static void PDM_EnableDMA (PDM_Type *base, bool enable)`  
*Enables/disables the PDM DMA requests.*
- `static uint32_t PDM_GetDataRegisterAddress (PDM_Type *base, uint32_t channel)`  
*Gets the PDM data register address.*

## Bus Operations

- `void PDM_ReadFifo (PDM_Type *base, uint32_t startChannel, uint32_t channelNums, void *buffer, size_t size, uint32_t dataWidth)`  
*PDM read fifo.*
- `static uint32_t PDM_ReadData (PDM_Type *base, uint32_t channel)`  
*Reads data from the PDM 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\\_hvvad\\_config\\_t](#) \*config)  
*Configure voice activity detector.*
- static void [PDM\\_ForceHvvadOutputDisable](#) (PDM\_Type \*base, bool enable)  
*PDM hvvad force output disable.*
- static void [PDM\\_ResetHvvad](#) (PDM\_Type \*base)  
*PDM hvvad 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 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.*

### 30.2.3 Data Structure Documentation

#### 30.2.3.1 struct pdm\_channel\_config\_t

##### Data Fields

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

#### 30.2.3.2 struct pdm\_config\_t

##### Data Fields

- bool enableDoze  
*This module will enter disable/low leakage mode if DOZEN is active with ipg\_doze is asserted.*
- uint8\_t fifoWatermark  
*Watermark value for FIFO.*
- `pdm_df_quality_mode_t` qualityMode  
*Quality mode.*
- uint8\_t cicOverSampleRate  
*CIC filter over sampling rate.*

#### 30.2.3.3 struct pdm\_hwvad\_config\_t

##### Data Fields

- uint8\_t channel  
*Which channel uses voice activity detector.*

- **uint8\_t initializeTime**  
*Number of frames or samples to initialize voice activity detector.*
- **uint8\_t cicOverSampleRate**  
*CIC filter over sampling rate.*
- **uint8\_t inputGain**  
*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\_t::initializeTime**

### 30.2.3.4 struct pdm\_hwvad\_noise\_filter\_t

#### Data Fields

- **bool enableAutoNoiseFilter**  
*If noise fileter automatically activated, true means enable.*
- **bool enableNoiseMin**  
*If Noise minimum block enabled, true means enabled.*
- **bool enableNoiseDecimation**  
*If enable noise input decimation.*
- **bool enableNoiseDetectOR**  
*Enables a OR logic in the output of minimum noise estimator block.*
- **uint32\_t noiseFilterAdjustment**  
*The adjustment value of the noise filter.*
- **uint32\_t noiseGain**  
*Gain value for the noise energy or envelope estimated.*

### 30.2.3.5 struct pdm\_hwvad\_zero\_cross\_detector\_t

#### Data Fields

- **bool enableAutoThreshold**  
*If ZCD auto-threshold enabled, true means enabled.*
- **pdm\_hwvad\_zcd\_result\_t zcdAnd**  
*Is ZCD result is AND'ed with energy-based detection, false means OR'ed.*
- **uint32\_t threshold**  
*The adjustment value of the noise filter.*
- **uint32\_t adjustmentThreshold**  
*Gain value for the noise energy or envelope estimated.*

**Field Documentation**(1) **bool pdm\_hwvad\_zero\_cross\_detector\_t::enableAutoThreshold****30.2.3.6 struct pdm\_transfer\_t****Data Fields**

- volatile uint8\_t \* **data**  
*Data start address to transfer.*
- volatile size\_t **dataSize**  
*Total Transfer bytes size.*

**Field Documentation**(1) **volatile uint8\_t\* pdm\_transfer\_t::data**(2) **volatile size\_t pdm\_transfer\_t::dataSize****30.2.3.7 struct pdm\_hwvad\_notification\_t****30.2.3.8 struct \_pdm\_handle**

PDM handle.

**Data Fields**

- uint32\_t **state**  
*Transfer status.*
- **pdm\_transfer\_callback\_t callback**  
*Callback function called at transfer event.*
- void \* **userData**  
*Callback parameter passed to callback function.*
- pdm\_transfer\_t **pdmQueue** [PDM\_XFER\_QUEUE\_SIZE]  
*Transfer queue storing queued transfer.*
- size\_t **transferSize** [PDM\_XFER\_QUEUE\_SIZE]  
*Data bytes need to transfer.*
- volatile uint8\_t **queueUser**  
*Index for user to queue transfer.*
- volatile uint8\_t **queueDriver**  
*Index for driver to get the transfer data and size.*
- uint32\_t **format**  
*data format*
- uint8\_t **watermark**  
*Watermark value.*
- uint8\_t **startChannel**  
*end channel*
- uint8\_t **channelNums**  
*Enabled channel number.*

### 30.2.4 Enumeration Type Documentation

#### 30.2.4.1 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.

#### 30.2.4.2 enum \_pdm\_interrupt\_enable

Enumerator

*kPDM\_ErrorInterruptEnable* PDM channel error interrupt enable.  
*kPDM\_FIFOInterruptEnable* PDM channel FIFO interrupt.

#### 30.2.4.3 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

#### 30.2.4.4 enum \_pdm\_channel\_enable\_mask

Enumerator

*kPDM\_EnableChannel0* channge1 0 enable mask

|                                   |                        |
|-----------------------------------|------------------------|
| <b><i>kPDM_EnableChannel1</i></b> | channge1 1 enable mask |
| <b><i>kPDM_EnableChannel2</i></b> | channge1 2 enable mask |
| <b><i>kPDM_EnableChannel3</i></b> | channge1 3 enable mask |
| <b><i>kPDM_EnableChannel4</i></b> | channge1 4 enable mask |
| <b><i>kPDM_EnableChannel5</i></b> | channge1 5 enable mask |
| <b><i>kPDM_EnableChannel6</i></b> | channge1 6 enable mask |
| <b><i>kPDM_EnableChannel7</i></b> | channge1 7 enable mask |

### 30.2.4.5 enum \_pdm\_fifo\_status

Enumerator

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

### 30.2.4.6 enum \_pdm\_range\_status

Enumerator

|                                            |                                 |
|--------------------------------------------|---------------------------------|
| <b><i>kPDM_RangeStatusUnderFlowCh0</i></b> | channel0 range status underflow |
| <b><i>kPDM_RangeStatusUnderFlowCh1</i></b> | channel1 range status underflow |
| <b><i>kPDM_RangeStatusUnderFlowCh2</i></b> | channel2 range status underflow |
| <b><i>kPDM_RangeStatusUnderFlowCh3</i></b> | channel3 range status underflow |
| <b><i>kPDM_RangeStatusUnderFlowCh4</i></b> | channel4 range status underflow |
| <b><i>kPDM_RangeStatusUnderFlowCh5</i></b> | channel5 range status underflow |
| <b><i>kPDM_RangeStatusUnderFlowCh6</i></b> | channel6 range status underflow |
| <b><i>kPDM_RangeStatusUnderFlowCh7</i></b> | channel7 range status underflow |
| <b><i>kPDM_RangeStatusOverFlowCh0</i></b>  | channel0 range status overflow  |
| <b><i>kPDM_RangeStatusOverFlowCh1</i></b>  | channel1 range status overflow  |

*kPDM\_RangeStatusOverFlowCh2* channel2 range status overflow  
*kPDM\_RangeStatusOverFlowCh3* channel3 range status overflow  
*kPDM\_RangeStatusOverFlowCh4* channel4 range status overflow  
*kPDM\_RangeStatusOverFlowCh5* channel5 range status overflow  
*kPDM\_RangeStatusOverFlowCh6* channel6 range status overflow  
*kPDM\_RangeStatusOverFlowCh7* channel7 range status overflow

### 30.2.4.7 enum pdm\_dc\_remover\_t

Enumerator

*kPDM\_DcRemoverCutOff21Hz* DC remover cut off 21HZ.  
*kPDM\_DcRemoverCutOff83Hz* DC remover cut off 83HZ.  
*kPDM\_DcRemoverCutOff152Hz* DC remover cut off 152HZ.  
*kPDM\_DcRemoverBypass* DC remover bypass.

### 30.2.4.8 enum pdm\_df\_quality\_mode\_t

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

### 30.2.4.9 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

### 30.2.4.10 enum pdm\_df\_output\_gain\_t

Enumerator

*kPDM\_DfOutputGain0* Decimation filter output gain 0.  
*kPDM\_DfOutputGain1* Decimation filter output gain 1.

*kPDM\_DfOutputGain2* Decimation filter output gain 2.  
*kPDM\_DfOutputGain3* Decimation filter output gain 3.  
*kPDM\_DfOutputGain4* Decimation filter output gain 4.  
*kPDM\_DfOutputGain5* Decimation filter output gain 5.  
*kPDM\_DfOutputGain6* Decimation filter output gain 6.  
*kPDM\_DfOutputGain7* Decimation filter output gain 7.  
*kPDM\_DfOutputGain8* Decimation filter output gain 8.  
*kPDM\_DfOutputGain9* Decimation filter output gain 9.  
*kPDM\_DfOutputGain10* Decimation filter output gain 10.  
*kPDM\_DfOutputGain11* Decimation filter output gain 11.  
*kPDM\_DfOutputGain12* Decimation filter output gain 12.  
*kPDM\_DfOutputGain13* Decimation filter output gain 13.  
*kPDM\_DfOutputGain14* Decimation filter output gain 14.  
*kPDM\_DfOutputGain15* Decimation filter output gain 15.

### 30.2.4.11 enum \_pdm\_data\_width

Enumerator

*kPDM\_DataWidth24* PDM data width 24bit.  
*kPDM\_DataWidth32* PDM data width 32bit.

### 30.2.4.12 enum \_pdm\_hwvad\_interrupt\_enable

Enumerator

*kPDM\_HwvadErrorInterruptEnable* PDM channel HWVAD error interrupt enable.  
*kPDM\_HwvadInterruptEnable* PDM channel HWVAD interrupt.

### 30.2.4.13 enum \_pdm\_hwvad\_int\_status

Enumerator

*kPDM\_HwvadStatusInputSaturation* HWVAD saturation condition.  
*kPDM\_HwvadStatusVoiceDetectFlag* HWVAD voice detect interrupt triggered.

### 30.2.4.14 enum pdm\_hwvad\_hpf\_config\_t

Enumerator

*kPDM\_HwvadHpfBypassed* High-pass filter bypass.  
*kPDM\_HwvadHpfCutOffFreq1750Hz* High-pass filter cut off frequency 1750HZ.

***kPDM\_HwvadHpfCutOffFreq215Hz*** High-pass filter cut off frequency 215HZ.  
***kPDM\_HwvadHpfCutOffFreq102Hz*** High-pass filter cut off frequency 102HZ.

### 30.2.4.15 enum pdm\_hwvad\_filter\_status\_t

Enumerator

***kPDM\_HwvadInternalFilterNormalOperation*** internal filter ready for normal operation  
***kPDM\_HwvadInternalFilterInitial*** interla filter are initial

### 30.2.4.16 enum pdm\_hwvad\_zcd\_result\_t

Enumerator

***kPDM\_HwvadResultOREnergyBasedDetection*** zero cross detector result will be OR with energy based detection  
***kPDM\_HwvadResultANDEnergyBasedDetection*** zero cross detector result will be AND with energy based detection

## 30.2.5 Function Documentation

### 30.2.5.1 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. |

### 30.2.5.2 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 |
|-------------|------------------|

### 30.2.5.3 static void PDM\_Reset ( PDM\_Type \* *base* ) [inline], [static]

Parameters

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

### 30.2.5.4 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. |

### 30.2.5.5 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. |

### 30.2.5.6 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. |

**30.2.5.7 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. |

**30.2.5.8 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. |

**30.2.5.9 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. |

**30.2.5.10 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. |

### 30.2.5.11 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.            |

### 30.2.5.12 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            |

### 30.2.5.13 `uint32_t PDM_GetInstance( PDM_Type * base )`

Parameters

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

### 30.2.5.14 `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.

### 30.2.5.15 `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.

### 30.2.5.16 `static uint32_t PDM_GetRangeStatus( PDM_Type * base ) [inline], [static]`

Use the Status Mask in \_pdm\_range\_status to get the status value needed

Parameters

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

Returns

output status.

### 30.2.5.17 static void PDM\_ClearStatus ( PDM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM base pointer                                                                                                        |
| <i>mask</i> | State mask. It can be a combination of the status between kPDM_StatusFrequency-Low and kPDM_StatusCh7FifoDataAvailable. |

### 30.2.5.18 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. |

### 30.2.5.19 static void PDM\_ClearRangeStatus ( 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_range_status. |

### 30.2.5.20 void PDM\_EnableInterrupts ( PDM\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM base pointer                                                                                                                                                                                           |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kPDM_ErrorInterruptEnable</li> <li>• kPDM_FIFOInterruptEnable</li> </ul> |

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

Parameters

|             |                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM base pointer                                                                                                                                                                                           |
| <i>mask</i> | interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> <li>• kPDM_ErrorInterruptEnable</li> <li>• kPDM_FIFOInterruptEnable</li> </ul> |

### 30.2.5.22 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. |

### 30.2.5.23 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.

### 30.2.5.24 void PDM\_ReadFifo ( PDM\_Type \* *base*, uint32\_t *startChannel*, uint32\_t *channelNums*, void \* *buffer*, size\_t *size*, uint32\_t *dataWidth* )

Note

: This function support 16 bit only for IP version that only supports 16bit.

Parameters

|                     |                            |
|---------------------|----------------------------|
| <i>base</i>         | PDM base pointer.          |
| <i>startChannel</i> | start channel number.      |
| <i>channelNums</i>  | total enabled channelnums. |
| <i>buffer</i>       | received buffer address.   |
| <i>size</i>         | number of samples to read. |
| <i>dataWidth</i>    | sample width.              |

### 30.2.5.25 static uint32\_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.

### 30.2.5.26 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. |

### 30.2.5.27 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 . |

### 30.2.5.28 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. |

### 30.2.5.29 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 |
|-------------|------------------|

### 30.2.5.30 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. |

### 30.2.5.31 static void PDM\_EnableHwvadInterrupts ( PDM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM 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>• kPDM_HWVADErrorInterruptEnable</li> <li>• kPDM_HWVADIInterruptEnable</li> </ul> |

### 30.2.5.32 static void PDM\_DisableHwvadInterrupts ( PDM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | PDM 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>• kPDM_HWVADErrorInterruptEnable</li> <li>• kPDM_HWVADIInterruptEnable</li> </ul> |

### 30.2.5.33 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. |

**30.2.5.34 static uint32\_t PDM\_GetHwvadInterruptStatusFlags ( PDM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

status, reference \_pdm\_hwvad\_int\_status

**30.2.5.35 static uint32\_t PDM\_GetHwvadInitialFlag ( PDM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

initial flag.

**30.2.5.36 static void PDM\_EnableHwvadSignalFilter ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                      |
| <i>enable</i> | True means enable signal filter, false means disable. |

**30.2.5.37 void PDM\_SetHwvadSignalFilterConfig ( PDM\_Type \* *base*, bool *enableMaxBlock*, uint32\_t *signalGain* )**

Parameters

|                        |                                   |
|------------------------|-----------------------------------|
| <i>base</i>            | PDM base pointer                  |
| <i>enableMax-Block</i> | If signal maximum block enabled.  |
| <i>signalGain</i>      | Gain value for the signal energy. |

**30.2.5.38 void PDM\_SetHwvadNoiseFilterConfig ( PDM\_Type \* *base*, const pdm\_hwvad\_noise\_filter\_t \* *config* )**

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                   |
| <i>config</i> | Voice activity detector noise filter configure structure pointer . |

**30.2.5.39 static void PDM\_EnableHwvadZeroCrossDetector ( PDM\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                            |
| <i>enable</i> | True means enable zero cross detector, false means disable. |

**30.2.5.40 void PDM\_SetHwvadZeroCrossDetectorConfig ( PDM\_Type \* *base*, const pdm\_hwvad\_zero\_cross\_detector\_t \* *config* )**

Parameters

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                          |
| <i>config</i> | Voice activity detector zero cross detector configure structure pointer . |

**30.2.5.41 static uint16\_t PDM\_GetNoiseData ( PDM\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

Data in PDM noise data register.

### 30.2.5.42 static void PDM\_SetHwvadInternalFilterStatus ( PDM\_Type \* *base*, pdm\_hwvad\_filter\_status\_t *status* ) [inline], [static]

Note: filter initial status should be asserted for two more cycles, then set it to normal operation.

Parameters

|               |                         |
|---------------|-------------------------|
| <i>base</i>   | PDM base pointer.       |
| <i>status</i> | internal filter status. |

### 30.2.5.43 void PDM\_SetHwvadInEnvelopeBasedMode ( PDM\_Type \* *base*, const pdm\_hwvad\_config\_t \* *hwvadConfig*, const pdm\_hwvad\_noise\_filter\_t \* *noiseConfig*, const pdm\_hwvad\_zero\_cross\_detector\_t \* *zcdConfig*, uint32\_t *signalGain* )

Recommand configurations,

```
* static const pdm_hwvad_config_t hwvadConfig = {
* .channel = 0,
* .initializeTime = 10U,
* .cicOverSampleRate = 0U,
* .inputGain = 0U,
* .frameTime = 10U,
* .cutOffFreq = kPDM_HwvadHpfBypassed,
* .enableFrameEnergy = false,
* .enablePreFilter = true,
};

* static const pdm_hwvad_noise_filter_t noiseFilterConfig = {
* .enableAutoNoiseFilter = false,
* .enableNoiseMin = true,
* .enableNoiseDecimation = true,
* .noiseFilterAdjustment = 0U,
* .noiseGain = 7U,
* .enableNoiseDetectOR = true,
* };
*
```

Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>base</i>        | PDM base pointer.                                                         |
| <i>hwvadConfig</i> | internal filter status.                                                   |
| <i>noiseConfig</i> | Voice activity detector noise filter configure structure pointer.         |
| <i>zcdConfig</i>   | Voice activity detector zero cross detector configure structure pointer . |
| <i>signalGain</i>  | signal gain value.                                                        |

**30.2.5.44 void PDM\_SetHwvadInEnergyBasedMode ( PDM\_Type \* *base*, const pdm\_hwvad\_config\_t \* *hwvadConfig*, const pdm\_hwvad\_noise\_filter\_t \* *noiseConfig*, const pdm\_hwvad\_zero\_cross\_detector\_t \* *zcdConfig*, uint32\_t *signalGain* )**

Recommand configurations, code static const [pdm\\_hwvad\\_config\\_t](#) hwvadConfig = { .channel = 0, .initializeTime = 10U, .cicOverSampleRate = 0U, .inputGain = 0U, .frameTime = 10U, .cutOffFreq = kPDM\_HwvadHpfBypassed, .enableFrameEnergy = true, .enablePreFilter = true, };

static const [pdm\\_hwvad\\_noise\\_filter\\_t](#) noiseFilterConfig = { .enableAutoNoiseFilter = true, .enableNoiseMin = false, .enableNoiseDecimation = false, .noiseFilterAdjustment = 0U, .noiseGain = 7U, .enableNoiseDetectOR = false, }; code param base PDM base pointer. param hwvadConfig internal filter status. param noiseConfig Voice activity detector noise filter configure structure pointer. param zcdConfig Voice activity detector zero cross detector configure structure pointer . param signalGain signal gain value, signal gain value should be properly according to application.

**30.2.5.45 void PDM\_EnableHwvadInterruptCallback ( PDM\_Type \* *base*, pdm\_hwvad\_callback\_t *vadCallback*, void \* *userData*, bool *enable* )**

This function enable/disable the hwvad interrupt for the selected PDM peripheral.

Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>base</i>        | Base address of the PDM peripheral.                                       |
| <i>vadCallback</i> | callback Pointer to store callback function, should be NULL when disable. |
| <i>userData</i>    | user data.                                                                |
| <i>enable</i>      | true is enable, false is disable.                                         |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

### 30.2.5.46 void PDM\_TransferCreateHandle ( **PDM\_Type** \* *base*, **pdm\_handle\_t** \* *handle*, **pdm\_transfer\_callback\_t** *callback*, **void** \* *userData* )

This function initializes the handle for the PDM transactional APIs. Call this function once to get the handle initialized.

Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>base</i>     | PDM base pointer.                               |
| <i>handle</i>   | PDM handle pointer.                             |
| <i>callback</i> | Pointer to the user callback function.          |
| <i>userData</i> | User parameter passed to the callback function. |

### 30.2.5.47 status\_t PDM\_TransferSetChannelConfig ( **PDM\_Type** \* *base*, **pdm\_handle\_t** \* *handle*, **uint32\_t** *channel*, **const pdm\_channel\_config\_t** \* *config*, **uint32\_t** *format* )

Parameters

|                |                                                                 |
|----------------|-----------------------------------------------------------------|
| <i>base</i>    | PDM base pointer.                                               |
| <i>handle</i>  | PDM handle pointer.                                             |
| <i>channel</i> | PDM channel.                                                    |
| <i>config</i>  | channel config.                                                 |
| <i>format</i>  | data format, support data width configurations,_pdm_data_width. |

Return values

|                                          |                     |
|------------------------------------------|---------------------|
| <i>kStatus_PDM_Channel-Config_Failed</i> | or kStatus_Success. |
|------------------------------------------|---------------------|

### 30.2.5.48 status\_t PDM\_TransferReceiveNonBlocking ( **PDM\_Type** \* *base*, **pdm\_handle\_t** \* *handle*, **pdm\_transfer\_t** \* *xfer* )

## Note

This API returns immediately after the transfer initiates. Call the PDM\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_PDM\_Busy, the transfer is finished.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                       |
| <i>handle</i> | Pointer to the pdm_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the 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.   |

**30.2.5.49 void PDM\_TransferAbortReceive ( PDM\_Type \* *base*, pdm\_handle\_t \* *handle* )**

## Note

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

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | PDM base pointer                                                       |
| <i>handle</i> | Pointer to the pdm_handle_t structure which stores the transfer state. |

**30.2.5.50 void PDM\_TransferHandleIRQ ( PDM\_Type \* *base*, pdm\_handle\_t \* *handle* )**

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | PDM base pointer.                      |
| <i>handle</i> | Pointer to the pdm_handle_t structure. |

# **Chapter 31**

## **PDM EDMA Driver**

# Chapter 32

## QTMR: Quad Timer Driver

### 32.1 Overview

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

### Data Structures

- struct `qtmr_config_t`  
*Quad Timer config structure. [More...](#)*

### Enumerations

- enum `qtmr_primary_count_source_t` {  
    `kQTMR_ClockCounter0InputPin` = 0,  
    `kQTMR_ClockCounter1InputPin`,  
    `kQTMR_ClockCounter2InputPin`,  
    `kQTMR_ClockCounter3InputPin`,  
    `kQTMR_ClockCounter0Output`,  
    `kQTMR_ClockCounter1Output`,  
    `kQTMR_ClockCounter2Output`,  
    `kQTMR_ClockCounter3Output`,  
    `kQTMR_ClockDivide_1`,  
    `kQTMR_ClockDivide_2`,  
    `kQTMR_ClockDivide_4`,  
    `kQTMR_ClockDivide_8`,  
    `kQTMR_ClockDivide_16`,  
    `kQTMR_ClockDivide_32`,  
    `kQTMR_ClockDivide_64`,  
    `kQTMR_ClockDivide_128` }  
*Quad Timer primary clock source selection.*
- enum `qtmr_input_source_t` {  
    `kQTMR_Counter0InputPin` = 0,  
    `kQTMR_Counter1InputPin`,  
    `kQTMR_Counter2InputPin`,  
    `kQTMR_Counter3InputPin` }  
*Quad Timer input sources selection.*
- enum `qtmr_counting_mode_t` {

```
kQTMR_NoOperation = 0,
kQTMR_PriSrcRiseEdge,
kQTMR_PriSrcRiseAndFallEdge,
kQTMR_PriSrcRiseEdgeSecInpHigh,
kQTMR_QuadCountMode,
kQTMR_PriSrcRiseEdgeSecDir,
kQTMR_SecSrcTrigPriCnt,
kQTMR_CascadeCount }
```

*Quad Timer counting mode selection.*

- enum `qtmr_pwm_out_state_t` {
 kQTMR\_PwmLow = 0,
 kQTMR\_PwmHigh }

*Quad Timer PWM output state.*

- enum `qtmr_output_mode_t` {
 kQTMR\_AssertWhenCountActive = 0,
 kQTMR\_ClearOnCompare,
 kQTMR\_SetOnCompare,
 kQTMR\_ToggleOnCompare,
 kQTMR\_ToggleOnAltCompareReg,
 kQTMR\_SetOnCompareClearOnSecSrcInp,
 kQTMR\_SetOnCompareClearOnCountRoll,
 kQTMR\_EnableGateClock }

*Quad Timer output mode selection.*

- enum `qtmr_input_capture_edge_t` {
 kQTMR\_NoCapture = 0,
 kQTMR\_RisingEdge,
 kQTMR\_FallingEdge,
 kQTMR\_RisingAndFallingEdge }

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum `qtmr_reload_control_t` {
 kQTMR\_NoPreload = 0,
 kQTMR\_LoadOnComp1,
 kQTMR\_LoadOnComp2 }

*Quad Timer input capture edge mode, rising edge, or falling edge.*

- enum `qtmr_debug_action_t` {
 kQTMR\_RunNormalInDebug = 0U,
 kQTMR\_HaltCounter,
 kQTMR\_ForceOutToZero,
 kQTMR\_HaltCountForceOutZero }

*List of Quad Timer run options when in Debug mode.*

- enum `qtmr_interrupt_enable_t` {
 kQTMR\_CompareInterruptEnable = (1U << 0),
 kQTMR\_Compare1InterruptEnable = (1U << 1),
 kQTMR\_Compare2InterruptEnable = (1U << 2),
 kQTMR\_OverflowInterruptEnable = (1U << 3),
 kQTMR\_EdgeInterruptEnable = (1U << 4) }

- *List of Quad Timer interrupts.*
- enum `qtmr_status_flags_t` {
   
    `kQTMR_CompareFlag` = (1U << 0),
   
    `kQTMR_Compare1Flag` = (1U << 1),
   
    `kQTMR_Compare2Flag` = (1U << 2),
   
    `kQTMR_OverflowFlag` = (1U << 3),
   
    `kQTMR_EdgeFlag` = (1U << 4) }
- List of Quad Timer flags.*
- enum `qtmr_channel_selection_t` {
   
    `kQTMR_Channel_0` = 0U,
   
    `kQTMR_Channel_1`,
   
    `kQTMR_Channel_2`,
   
    `kQTMR_Channel_3` }
- List of channel selection.*
- enum `qtmr_dma_enable_t` {
   
    `kQTMR_InputEdgeFlagDmaEnable` = (1U << 0),
   
    `kQTMR_ComparatorPreload1DmaEnable` = (1U << 1),
   
    `kQTMR_ComparatorPreload2DmaEnable` = (1U << 2) }
- List of Quad Timer DMA enable.*

## Functions

- `status_t QTMR_SetupPwm (TMR_Type *base, qtmr_channel_selection_t channel, uint32_t pwmFreqHz, uint8_t dutyCyclePercent, bool outputPolarity, uint32_t srcClock_Hz)`
  
*Sets up Quad timer module for PWM signal output.*
- void `QTMR_SetupInputCapture (TMR_Type *base, qtmr_channel_selection_t channel, qtmr_input_source_t capturePin, bool inputPolarity, bool reloadOnCapture, qtmr_input_capture_edge_t captureMode)`
  
*Allows the user to count the source clock cycles until a capture event arrives.*

## Driver version

- #define `FSL_QTMR_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))`
  
*Version.*

## Initialization and deinitialization

- void `QTMR_Init (TMR_Type *base, qtmr_channel_selection_t channel, const qtmr_config_t *config)`
  
*Ungates the Quad Timer clock and configures the peripheral for basic operation.*
- void `QTMR_Deinit (TMR_Type *base, qtmr_channel_selection_t channel)`
  
*Stops the counter and gates the Quad Timer clock.*
- void `QTMR_GetDefaultConfig (qtmr_config_t *config)`
  
*Fill in the Quad Timer config struct with the default settings.*

## Interrupt Interface

- void `QTMR_EnableInterrupts (TMR_Type *base, qtmr_channel_selection_t channel, uint32_t mask)`

- void `QTMR_DisableInterrupts` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, uint32\_t mask)  
*Enables the selected Quad Timer interrupts.*
- uint32\_t `QTMR_GetEnabledInterrupts` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel)  
*Disables the selected Quad Timer interrupts.*  
*Gets the enabled Quad Timer interrupts.*

## Status Interface

- uint32\_t `QTMR_GetStatus` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel)  
*Gets the Quad Timer status flags.*
- void `QTMR_ClearStatusFlags` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, uint32\_t mask)  
*Clears the Quad Timer status flags.*

## Read and Write the timer period

- void `QTMR_SetTimerPeriod` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, uint16\_t ticks)  
*Sets the timer period in ticks.*
- void `QTMR_SetCompareValue` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, uint16\_t ticks)  
*Set compare value.*
- static void `QTMR_SetLoadValue` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, uint16\_t value)  
*Set load value.*
- static uint16\_t `QTMR_GetCurrentTimerCount` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void `QTMR_StartTimer` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, qtmr\_counting\_mode\_t clockSource)  
*Starts the Quad Timer counter.*
- static void `QTMR_StopTimer` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel)  
*Stops the Quad Timer counter.*

## Enable and Disable the Quad Timer DMA

- void `QTMR_EnableDma` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, uint32\_t mask)  
*Enable the Quad Timer DMA.*
- void `QTMR_DisableDma` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, uint32\_t mask)  
*Disable the Quad Timer DMA.*
- void `QTMR_SetPwmOutputToIdle` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel, bool idleStatus)  
*Set PWM output in idle status (high or low).*
- static qtmr\_pwm\_out\_state\_t `QTMR_GetPwmOutputStatus` (TMR\_Type \*base, qtmr\_channel\_selection\_t channel)  
*Get the channel output status.*

- `uint8_t QTMR_GetPwmChannelStatus (TMR_Type *base, qtmr_channel_selection_t channel)`  
*Get the PWM channel dutycycle value.*
- `void QTMR_SetPwmClockMode (TMR_Type *base, qtmr_channel_selection_t channel, qtmr_primary_count_source_t prescaler)`  
*This function set the value of the prescaler on QTimer channels.*

## 32.2 Data Structure Documentation

### 32.2.1 struct qtmr\_config\_t

This structure holds the configuration settings for the Quad Timer peripheral. To initialize this structure to reasonable defaults, call the `QTMR_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

### Data Fields

- `qtmr_primary_count_source_t primarySource`  
*Specify the primary count source.*
- `qtmr_input_source_t secondarySource`  
*Specify the secondary count source.*
- `bool enableMasterMode`  
*true: Broadcast compare function output to other counters; false no broadcast*
- `bool enableExternalForce`  
*true: Compare from another counter force state of OFLAG signal false: OFLAG controlled by local counter*
- `uint8_t faultFilterCount`  
*Fault filter count.*
- `uint8_t faultFilterPeriod`  
*Fault filter period; value of 0 will bypass the filter.*
- `qtmr_debug_action_t debugMode`  
*Operation in Debug mode.*

## 32.3 Enumeration Type Documentation

### 32.3.1 enum qtmr\_primary\_count\_source\_t

Enumerator

- |                                          |                          |
|------------------------------------------|--------------------------|
| <code>kQTMR_ClockCounter0InputPin</code> | Use counter 0 input pin. |
| <code>kQTMR_ClockCounter1InputPin</code> | Use counter 1 input pin. |
| <code>kQTMR_ClockCounter2InputPin</code> | Use counter 2 input pin. |
| <code>kQTMR_ClockCounter3InputPin</code> | Use counter 3 input pin. |
| <code>kQTMR_ClockCounter0Output</code>   | Use counter 0 output.    |
| <code>kQTMR_ClockCounter1Output</code>   | Use counter 1 output.    |
| <code>kQTMR_ClockCounter2Output</code>   | Use counter 2 output.    |
| <code>kQTMR_ClockCounter3Output</code>   | Use counter 3 output.    |

*kQTMR\_ClockDivide\_1* IP bus clock divide by 1 prescaler.  
*kQTMR\_ClockDivide\_2* IP bus clock divide by 2 prescaler.  
*kQTMR\_ClockDivide\_4* IP bus clock divide by 4 prescaler.  
*kQTMR\_ClockDivide\_8* IP bus clock divide by 8 prescaler.  
*kQTMR\_ClockDivide\_16* IP bus clock divide by 16 prescaler.  
*kQTMR\_ClockDivide\_32* IP bus clock divide by 32 prescaler.  
*kQTMR\_ClockDivide\_64* IP bus clock divide by 64 prescaler.  
*kQTMR\_ClockDivide\_128* IP bus clock divide by 128 prescaler.

### 32.3.2 enum qtmr\_input\_source\_t

Enumerator

*kQTMR\_Counter0InputPin* Use counter 0 input pin.  
*kQTMR\_Counter1InputPin* Use counter 1 input pin.  
*kQTMR\_Counter2InputPin* Use counter 2 input pin.  
*kQTMR\_Counter3InputPin* Use counter 3 input pin.

### 32.3.3 enum qtmr\_counting\_mode\_t

Enumerator

*kQTMR\_NoOperation* No operation.  
*kQTMR\_PriSrcRiseEdge* Count rising edges of primary source.  
*kQTMR\_PriSrcRiseAndFallEdge* Count rising and falling edges of primary source.  
*kQTMR\_PriSrcRiseEdgeSecInpHigh* Count rise edges of pri SRC while sec inp high active.  
*kQTMR\_QuadCountMode* Quadrature count mode, uses pri and sec sources.  
*kQTMR\_PriSrcRiseEdgeSecDir* Count rising edges of pri SRC; sec SRC specifies dir.  
*kQTMR\_SecSrcTrigPriCnt* Edge of sec SRC trigger primary count until compare.  
*kQTMR\_CascadeCount* Cascaded count mode (up/down)

### 32.3.4 enum qtmr\_pwm\_out\_state\_t

Enumerator

*kQTMR\_PwmLow* The output state of PWM channel is low.  
*kQTMR\_PwmHigh* The output state of PWM channel is low.

### 32.3.5 enum qtmr\_output\_mode\_t

Enumerator

- kQTMR AssertWhenCountActive* Assert OFLAG while counter is active.
- kQTMR ClearOnCompare* Clear OFLAG on successful compare.
- kQTMR SetOnCompare* Set OFLAG on successful compare.
- kQTMR ToggleOnCompare* Toggle OFLAG on successful compare.
- kQTMR ToggleOnAltCompareReg* Toggle OFLAG using alternating compare registers.
- kQTMR SetOnCompareClearOnSecSrcInp* Set OFLAG on compare, clear on sec SRC input edge.
  
- kQTMR SetOnCompareClearOnCountRoll* Set OFLAG on compare, clear on counter rollover.
- kQTMR EnableGateClock* Enable gated clock output while count is active.

### 32.3.6 enum qtmr\_input\_capture\_edge\_t

Enumerator

- kQTMR NoCapture* Capture is disabled.
- kQTMR RisingEdge* Capture on rising edge (IPS=0) or falling edge (IPS=1)
- kQTMR FallingEdge* Capture on falling edge (IPS=0) or rising edge (IPS=1)
- kQTMR RisingAndFallingEdge* Capture on both edges.

### 32.3.7 enum qtmr\_reload\_control\_t

Enumerator

- kQTMR NoPreload* Never preload.
- kQTMR LoadOnComp1* Load upon successful compare with value in COMP1.
- kQTMR LoadOnComp2* Load upon successful compare with value in COMP2.

### 32.3.8 enum qtmr\_debug\_action\_t

Enumerator

- kQTMR RunNormalInDebug* Continue with normal operation.
- kQTMR HaltCounter* Halt counter.
- kQTMR ForceOutToZero* Force output to logic 0.
- kQTMR HaltCountForceOutZero* Halt counter and force output to logic 0.

### 32.3.9 enum qtmr\_interrupt\_enable\_t

Enumerator

- kQTMR\_CompareInterruptEnable* Compare interrupt.
- kQTMR\_Compare1InterruptEnable* Compare 1 interrupt.
- kQTMR\_Compare2InterruptEnable* Compare 2 interrupt.
- kQTMR\_OverflowInterruptEnable* Timer overflow interrupt.
- kQTMR\_EdgeInterruptEnable* Input edge interrupt.

### 32.3.10 enum qtmr\_status\_flags\_t

Enumerator

- kQTMR\_CompareFlag* Compare flag.
- kQTMR\_Compare1Flag* Compare 1 flag.
- kQTMR\_Compare2Flag* Compare 2 flag.
- kQTMR\_OverflowFlag* Timer overflow flag.
- kQTMR\_EdgeFlag* Input edge flag.

### 32.3.11 enum qtmr\_channel\_selection\_t

Enumerator

- kQTMR\_Channel\_0* TMR Channel 0.
- kQTMR\_Channel\_1* TMR Channel 1.
- kQTMR\_Channel\_2* TMR Channel 2.
- kQTMR\_Channel\_3* TMR Channel 3.

### 32.3.12 enum qtmr\_dma\_enable\_t

Enumerator

- kQTMR\_InputEdgeFlagDmaEnable* Input Edge Flag DMA Enable.
- kQTMR\_ComparatorPreload1DmaEnable* Comparator Preload Register 1 DMA Enable.
- kQTMR\_ComparatorPreload2DmaEnable* Comparator Preload Register 2 DMA Enable.

## 32.4 Function Documentation

### 32.4.1 void QTMR\_Init ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, const qtmr\_config\_t \* *config* )

## Note

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

## Parameters

|                |                                               |
|----------------|-----------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address            |
| <i>channel</i> | Quad Timer channel number                     |
| <i>config</i>  | Pointer to user's Quad Timer config structure |

**32.4.2 void QTMR\_Deinit ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* )**

## Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

**32.4.3 void QTMR\_GetDefaultConfig ( qtmr\_config\_t \* *config* )**

The default values are:

```
* config->debugMode = kQTMR_RunNormalInDebug;
* config->enableExternalForce = false;
* config->enableMasterMode = false;
* config->faultFilterCount = 0;
* config->faultFilterPeriod = 0;
* config->primarySource = kQTMR_ClockDivide_2;
* config->secondarySource = kQTMR_Counter0InputPin;
*
```

## Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>config</i> | Pointer to user's Quad Timer config structure. |
|---------------|------------------------------------------------|

**32.4.4 status\_t QTMR\_SetupPwm ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *pwmFreqHz*, uint8\_t *dutyCyclePercent*, bool *outputPolarity*, uint32\_t *srcClock\_Hz* )**

The function initializes the timer module according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements.

Parameters

|                          |                                                                                                                           |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | Quad Timer peripheral base address                                                                                        |
| <i>channel</i>           | Quad Timer channel number                                                                                                 |
| <i>pwmFreqHz</i>         | PWM signal frequency in Hz                                                                                                |
| <i>dutyCycle-Percent</i> | PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |
| <i>outputPolarity</i>    | true: invert polarity of the output signal, false: no inversion                                                           |
| <i>srcClock_Hz</i>       | Main counter clock in Hz.                                                                                                 |

Returns

Returns an error if there was error setting up the signal.

### 32.4.5 void QTMR\_SetupInputCapture ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, qtmr\_input\_source\_t *capturePin*, bool *inputPolarity*, bool *reloadOnCapture*, qtmr\_input\_capture\_edge\_t *captureMode* )

The count is stored in the capture register.

Parameters

|                         |                                                                         |
|-------------------------|-------------------------------------------------------------------------|
| <i>base</i>             | Quad Timer peripheral base address                                      |
| <i>channel</i>          | Quad Timer channel number                                               |
| <i>capturePin</i>       | Pin through which we receive the input signal to trigger the capture    |
| <i>inputPolarity</i>    | true: invert polarity of the input signal, false: no inversion          |
| <i>reloadOn-Capture</i> | true: reload the counter when an input capture occurs, false: no reload |
| <i>captureMode</i>      | Specifies which edge of the input signal triggers a capture             |

### 32.4.6 void QTMR\_EnableInterrupts ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )

Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                                   |
| <i>channel</i> | Quad Timer channel number                                                                                            |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a> |

### 32.4.7 void QTMR\_DisableInterrupts ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )

Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                                   |
| <i>channel</i> | Quad Timer channel number                                                                                            |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_interrupt_enable_t</a> |

### 32.4.8 uint32\_t QTMR\_GetEnabledInterrupts ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* )

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

Returns

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

### 32.4.9 uint32\_t QTMR\_GetStatus ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* )

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

Returns

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

### **32.4.10 void QTMR\_ClearStatusFlags ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                                |
| <i>channel</i> | Quad Timer channel number                                                                                         |
| <i>mask</i>    | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">qtmr_status_flags_t</a> |

### **32.4.11 void QTMR\_SetTimerPeriod ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint16\_t *ticks* )**

Timers counts from initial value till it equals the count value set here. The counter will then reinitialize to the value specified in the Load register.

Note

1. This function will write the time period in ticks to COMP1 or COMP2 register depending on the count direction
2. User can call the utility macros provided in fsl\_common.h to convert to ticks
3. This function supports cases, providing only primary source clock without secondary source clock.

Parameters

---

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |
| <i>ticks</i>   | Timer period in units of ticks     |

### 32.4.12 void QTMR\_SetCompareValue ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint16\_t *ticks* )

This function sets the value used for comparison with the counter value.

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |
| <i>ticks</i>   | Timer period in units of ticks.    |

### 32.4.13 static void QTMR\_SetLoadValue ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint16\_t *value* ) [inline], [static]

This function sets the value used to initialize the counter after a counter comparison.

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address  |
| <i>channel</i> | Quad Timer channel number           |
| <i>value</i>   | Load register initialization value. |

### 32.4.14 static uint16\_t QTMR\_GetCurrentTimerCount ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in fsl\_common.h to convert ticks to usec or msec

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

Returns

Current counter value in ticks

**32.4.15 static void QTMR\_StartTimer ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, qtmr\_counting\_mode\_t *clockSource* ) [inline], [static]**

Parameters

|                    |                                    |
|--------------------|------------------------------------|
| <i>base</i>        | Quad Timer peripheral base address |
| <i>channel</i>     | Quad Timer channel number          |
| <i>clockSource</i> | Quad Timer clock source            |

**32.4.16 static void QTMR\_StopTimer ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* ) [inline], [static]**

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

**32.4.17 void QTMR\_EnableDma ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )**

Parameters

---

|                |                                                                                                         |
|----------------|---------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                      |
| <i>channel</i> | Quad Timer channel number                                                                               |
| <i>mask</i>    | The DMA to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_dma_enable_t</a> |

### 32.4.18 void QTMR\_DisableDma ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, uint32\_t *mask* )

Parameters

|                |                                                                                                         |
|----------------|---------------------------------------------------------------------------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address                                                                      |
| <i>channel</i> | Quad Timer channel number                                                                               |
| <i>mask</i>    | The DMA to enable. This is a logical OR of members of the enumeration <a href="#">qtmr_dma_enable_t</a> |

### 32.4.19 void QTMR\_SetPwmOutputToldle ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, bool *idleStatus* )

Note

When the PWM is set again, the counting needs to be restarted.

Parameters

|                   |                                                                                   |
|-------------------|-----------------------------------------------------------------------------------|
| <i>base</i>       | Quad Timer peripheral base address                                                |
| <i>channel</i>    | Quad Timer channel number                                                         |
| <i>idleStatus</i> | True: PWM output is high in idle status; false: PWM output is low in idle status. |

### 32.4.20 static qtmr\_pwm\_out\_state\_t QTMR\_GetPwmOutputStatus ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* ) [inline], [static]

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

Returns

Current channel output status.

### 32.4.21 **uint8\_t QTMR\_GetPwmChannelStatus ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel* )**

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>base</i>    | Quad Timer peripheral base address |
| <i>channel</i> | Quad Timer channel number          |

Returns

Current channel dutycycle value.

### 32.4.22 **void QTMR\_SetPwmClockMode ( TMR\_Type \* *base*, qtmr\_channel\_selection\_t *channel*, qtmr\_primary\_count\_source\_t *prescaler* )**

Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>base</i>      | Quad Timer peripheral base address |
| <i>channel</i>   | Quad Timer channel number          |
| <i>prescaler</i> | Set prescaler value                |

# Chapter 33

## RGPIO: Rapid General-Purpose Input/Output Driver

### 33.1 Overview

#### Modules

- FGPIO Driver
- RGPIO Driver

#### Data Structures

- struct `rgpio_pin_config_t`  
*The RGPIO pin configuration structure.* [More...](#)

#### Enumerations

- enum `rgpio_pin_direction_t` {  
  `kRGPIO_DigitalInput` = 0U,  
  `kRGPIO_DigitalOutput` = 1U }  
*RGPIO direction definition.*
- enum `rgpio_interrupt_sel_t` {  
  `kRGPIO_InterruptOutput0` = 0x0U,  
  `kRGPIO_InterruptOutput1` = 0x1U,  
  `kRGPIO_InterruptOutput2` = 0x2U,  
  `kRGPIO_InterruptOutput3` = 0x3U }  
*Configures the interrupt generation condition.*
- enum `rgpio_interrupt_config_t` {  
  `kRGPIO_InterruptOrDMADisabled` = 0x0U,  
  `kRGPIO_DMARisingEdge` = 0x1U,  
  `kRGPIO_DMAFallingEdge` = 0x2U,  
  `kRGPIO_DMAEitherEdge` = 0x3U,  
  `kRGPIO_FlagRisingEdge` = 0x05U,  
  `kRGPIO_FlagFallingEdge` = 0x06U,  
  `kRGPIO_FlagEitherEdge` = 0x07U,  
  `kRGPIO_InterruptLogicZero` = 0x8U,  
  `kRGPIO_InterruptRisingEdge` = 0x9U,  
  `kRGPIO_InterruptFallingEdge` = 0xAU,  
  `kRGPIO_InterruptEitherEdge` = 0xBU,  
  `kRGPIO_InterruptLogicOne` = 0xCU,  
  `kRGPIO_ActiveHighTriggerOutputEnable` = 0xDU,  
  `kRGPIO_ActiveLowTriggerOutputEnable` = 0xEU }  
*Configures the interrupt generation condition.*

## Driver version

- #define **FSL\_RGPIO\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 1, 0))  
*RGPIO driver version 2.1.0.*

## 33.2 Data Structure Documentation

### 33.2.1 struct rgpio\_pin\_config\_t

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT\_SetPinConfig().

#### Data Fields

- **rgpio\_pin\_direction\_t pinDirection**  
*RGPIO direction, input or output.*
- **uint8\_t outputLogic**  
*Set a default output logic, which has no use in input.*

## 33.3 Macro Definition Documentation

### 33.3.1 #define FSL\_RGPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

## 33.4 Enumeration Type Documentation

### 33.4.1 enum rgpio\_pin\_direction\_t

Enumerator

**kRGPIO\_DigitalInput** Set current pin as digital input.

**kRGPIO\_DigitalOutput** Set current pin as digital output.

### 33.4.2 enum rgpio\_interrupt\_sel\_t

Enumerator

**kRGPIO\_InterruptOutput0** Interrupt/DMA request/trigger output 0.

**kRGPIO\_InterruptOutput1** Interrupt/DMA request/trigger output 1.

**kRGPIO\_InterruptOutput2** Interrupt/DMA request/trigger output 2.

**kRGPIO\_InterruptOutput3** Interrupt/DMA request/trigger output 3.

### 33.4.3 enum rgpio\_interrupt\_config\_t

Enumerator

- kRGPIO\_InterruptOrDMADisabled* Interrupt/DMA request is disabled.
- kRGPIO\_DMARisingEdge* DMA request on rising edge.
- kRGPIO\_DMAFallingEdge* DMA request on falling edge.
- kRGPIO\_DMAEitherEdge* DMA request on either edge.
- kRGPIO\_FlagRisingEdge* Flag sets on rising edge.
- kRGPIO\_FlagFallingEdge* Flag sets on falling edge.
- kRGPIO\_FlagEitherEdge* Flag sets on either edge.
- kRGPIO\_InterruptLogicZero* Interrupt when logic zero.
- kRGPIO\_InterruptRisingEdge* Interrupt on rising edge.
- kRGPIO\_InterruptFallingEdge* Interrupt on falling edge.
- kRGPIO\_InterruptEitherEdge* Interrupt on either edge.
- kRGPIO\_InterruptLogicOne* Interrupt when logic one.
- kRGPIO\_ActiveHighTriggerOutputEnable* Enable active high-trigger output.
- kRGPIO\_ActiveLowTriggerOutputEnable* Enable active low-trigger output.

## 33.5 RGPIO Driver

### 33.5.1 Overview

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

### 33.5.2 Typical use case

#### 33.5.2.1 Output Operation

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

#### 33.5.2.2 Input Operation

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

## Functions

- static void [RGPIO\\_SetPinInterruptConfig](#) (RGPIO\_Type \*base, uint32\_t pin, [rgpio\\_interrupt\\_sel\\_t](#) sel, [rgpio\\_interrupt\\_config\\_t](#) config)  
*Configures the gpio pin interrupt/DMA request.*
- static void [\\_SetMultipleInterruptPinsConfig](#) (RGPIO\_Type \*base, uint32\_t mask, [rgpio\\_interrupt\\_sel\\_t](#) sel, [rgpio\\_interrupt\\_config\\_t](#) config)  
*Sets the gpio interrupt configuration in ICR register for multiple pins.*
- static uint32\_t [RGPIO\\_GetPinsInterruptFlags](#) (RGPIO\_Type \*base, [rgpio\\_interrupt\\_sel\\_t](#) sel)  
*Reads the whole gpio status flag.*
- static void [RGPIO\\_ClearPinsInterruptFlags](#) (RGPIO\_Type \*base, [rgpio\\_interrupt\\_sel\\_t](#) sel, uint32\_t mask)  
*Clears the multiple pin interrupt status flag.*

## RGPIO Configuration

- void [RGPIO\\_PinInit](#) (RGPIO\_Type \*base, uint32\_t pin, const [rgpio\\_pin\\_config\\_t](#) \*config)  
*Initializes a RGPIO pin used by the board.*
- uint32\_t [RGPIO.GetInstance](#) (RGPIO\_Type \*base)  
*Gets the RGPIO instance according to the RGPIO base.*

## RGPIO Output Operations

- static void [RGPIO\\_PinWrite](#) (RGPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the multiple RGPIO pins to the logic 1 or 0.*
- static void [RGPIO\\_WritePinOutput](#) (RGPIO\_Type \*base, uint32\_t pin, uint8\_t output)

- static void **GPIO\_PortSet** (GPIO\_Type \*base, uint32\_t mask)
 

*Sets the output level of the multiple GPIO pins to the logic 1 or 0.*
- static void **GPIO\_SetPinsOutput** (GPIO\_Type \*base, uint32\_t mask)
 

*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void **GPIO\_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 void **GPIO\_TogglePinsOutput** (GPIO\_Type \*base, uint32\_t mask)
 

*Reverses the current output logic of the multiple GPIO pins.*

## GPIO Input Operations

- 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.*
- static void **GPIO\_EnablePortInput** (GPIO\_Type \*base, uint32\_t mask, bool enable)

### 33.5.3 Function Documentation

#### 33.5.3.1 void **GPIO\_PinInit** ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **const rgpio\_pin\_config\_t** \* *config* )

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the **GPIO\_PinInit()** function.

This is an example to define an input pin or an output pin configuration.

```
* Define a digital input pin configuration,
* rgpio_pin_config_t config =
* {
* kRGPIO_DigitalInput,
* 0,
* }
* Define a digital output pin configuration,
* rgpio_pin_config_t config =
* {
* kRGPIO_DigitalOutput,
* 0,
* }
```

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>pin</i>    | GPIO port pin number                                           |
| <i>config</i> | GPIO pin configuration pointer                                 |

### 33.5.3.2 `uint32_t GPIO_GetInstance ( GPIO_Type * base )`

Parameters

|             |                                                   |
|-------------|---------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(PTA, PTB, PTC, etc.) |
|-------------|---------------------------------------------------|

Return values

|             |          |
|-------------|----------|
| <i>GPIO</i> | instance |
|-------------|----------|

### 33.5.3.3 `static void GPIO_PinWrite ( GPIO_Type * base, uint32_t pin, uint8_t output ) [inline], [static]`

Parameters

|               |                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)                                                                                                                         |
| <i>pin</i>    | GPIO pin number                                                                                                                                                                        |
| <i>output</i> | GPIO pin output logic level. <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul> |

### 33.5.3.4 `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](#).

### 33.5.3.5 `static void GPIO_PortSet ( GPIO_Type * base, uint32_t mask ) [inline], [static]`

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.) |
| <i>mask</i> | RGPIO pin number macro                                             |

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

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

### 33.5.3.7 static void RGPIO\_PortClear ( RGPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.) |
| <i>mask</i> | RGPIO pin number macro                                             |

### 33.5.3.8 static void RGPIO\_ClearPinsOutput ( RGPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

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

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.) |
| <i>mask</i> | RGPIO pin number macro                                             |

### 33.5.3.9 static void RGPIO\_PortToggle ( RGPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

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

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

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

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>pin</i>  | GPIO pin number                                                |

Return values

|             |                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>GPIO</i> | port input value <ul style="list-style-type: none"> <li>• 0: corresponding pin input low-logic level.</li> <li>• 1: corresponding pin input high-logic level.</li> </ul> |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**33.5.3.12 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](#).

**33.5.3.13 static void GPIO\_EnablePortInput ( GPIO\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

Parameters

---

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i>   | GPIO pin number mask                                           |
| <i>enable</i> | GPIO digital input enable/disable flag.                        |

**33.5.3.14 static void GPIO\_SetPinInterruptConfig ( **GPIO\_Type** \* *base*, **uint32\_t** *pin*, **rgpio\_interrupt\_sel\_t** *sel*, **rgpio\_interrupt\_config\_t** *config* ) [inline], [static]**

Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer.      |
| <i>pin</i>    | GPIO pin number.                   |
| <i>sel</i>    | GPIO pin interrupt selection(0-3). |
| <i>config</i> | GPIO pin interrupt configuration.  |

**33.5.3.15 static void \_SetMultipleInterruptPinsConfig ( **GPIO\_Type** \* *base*, **uint32\_t** *mask*, **rgpio\_interrupt\_sel\_t** *sel*, **rgpio\_interrupt\_config\_t** *config* ) [inline], [static]**

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i>   | GPIO pin number macro.                                         |
| <i>sel</i>    | GPIO pin interrupt selection(0-3).                             |
| <i>config</i> | GPIO pin interrupt configuration.                              |

**33.5.3.16 static uint32\_t GPIO\_GetPinsInterruptFlags ( **GPIO\_Type** \* *base*, **rgpio\_interrupt\_sel\_t** *sel* ) [inline], [static]**

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | RGPIO peripheral base pointer.      |
| <i>sel</i>  | RGPIO pin interrupt selection(0-3). |

Returns

Current gpio interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

### 33.5.3.17 static void RGPIO\_ClearPinsInterruptFlags ( RGPIO\_Type \* *base*, rgpio\_interrupt\_sel\_t *sel*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | RGPIO peripheral base pointer.      |
| <i>sel</i>  | RGPIO pin interrupt selection(0-3). |
| <i>mask</i> | RGPIO pin number macro.             |

## 33.6 FGPIO Driver

This section describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the RGPI0 application.

Note

FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and RGPI0 share the same peripheral but use different registers. FGPIO is closer to the core than the regular RGPI0 and it's faster to read and write.

### 33.6.1 Typical use case

#### 33.6.1.1 Output Operation

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

#### 33.6.1.2 Input Operation

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

# Chapter 34

## RTWDOG: 32-bit Watchdog Timer

### 34.1 Overview

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

### 34.2 Typical use case

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

## Data Structures

- struct `rtwdog_work_mode_t`  
*Defines RTWDOG work mode. [More...](#)*
- struct `rtwdog_config_t`  
*Describes RTWDOG configuration structure. [More...](#)*

## Enumerations

- enum `rtwdog_clock_source_t` {  
  kRTWDOG\_ClockSource0 = 0U,  
  kRTWDOG\_ClockSource1 = 1U,  
  kRTWDOG\_ClockSource2 = 2U,  
  kRTWDOG\_ClockSource3 = 3U }  
*Describes RTWDOG clock source.*
- enum `rtwdog_clock_prescaler_t` {  
  kRTWDOG\_ClockPrescalerDivide1 = 0x0U,  
  kRTWDOG\_ClockPrescalerDivide256 = 0x1U }  
*Describes the selection of the clock prescaler.*
- enum `rtwdog_test_mode_t` {  
  kRTWDOG\_TestModeDisabled = 0U,  
  kRTWDOG\_UserModeEnabled = 1U,  
  kRTWDOG\_LowByteTest = 2U,  
  kRTWDOG\_HighByteTest = 3U }  
*Describes RTWDOG test mode.*
- enum `_rtwdog_interrupt_enable_t` { kRTWDOG\_InterruptEnable = RTWDOG\_CS\_INT\_MASK }  
*RTWDOG interrupt configuration structure.*
- enum `_rtwdog_status_flags_t` {  
  kRTWDOG\_RunningFlag = RTWDOG\_CS\_EN\_MASK,  
  kRTWDOG\_InterruptFlag = RTWDOG\_CS\_FLG\_MASK }  
*RTWDOG status flags.*

## Unlock sequence

- #define **WDOG\_FIRST\_WORD\_OF\_UNLOCK** (RTWDOG\_UPDATE\_KEY & 0xFFFFU)  
*First word of unlock sequence.*
- #define **WDOG\_SECOND\_WORD\_OF\_UNLOCK** ((RTWDOG\_UPDATE\_KEY >> 16U) & 0xFFFFU)  
*Second word of unlock sequence.*

## Refresh sequence

- #define **WDOG\_FIRST\_WORD\_OF\_REFRESH** (RTWDOG\_REFRESH\_KEY & 0xFFFFU)  
*First word of refresh sequence.*
- #define **WDOG\_SECOND\_WORD\_OF\_REFRESH** ((RTWDOG\_REFRESH\_KEY >> 16U) & 0xFFFFU)  
*Second word of refresh sequence.*

## Driver version

- #define **FSL\_RTWDOG\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 2))  
*RTWDOG driver version 2.1.2.*

## RTWDOG Initialization and De-initialization

- void **RTWDOG\_GetDefaultConfig** (rtwdog\_config\_t \*config)  
*Initializes the RTWDOG configuration structure.*
- void **RTWDOG\_Init** (RTWDOG\_Type \*base, const rtwdog\_config\_t \*config)  
*Initializes the RTWDOG module.*
- void **RTWDOG\_Deinit** (RTWDOG\_Type \*base)  
*De-initializes the RTWDOG module.*

## RTWDOG functional Operation

- static void **RTWDOG\_Enable** (RTWDOG\_Type \*base)  
*Enables the RTWDOG module.*
- static void **RTWDOG\_Disable** (RTWDOG\_Type \*base)  
*Disables the RTWDOG module.*
- static void **RTWDOG\_EnableInterrupts** (RTWDOG\_Type \*base, uint32\_t mask)  
*Enables the RTWDOG interrupt.*
- static void **RTWDOG\_DisableInterrupts** (RTWDOG\_Type \*base, uint32\_t mask)  
*Disables the RTWDOG interrupt.*
- static uint32\_t **RTWDOG\_GetStatusFlags** (RTWDOG\_Type \*base)  
*Gets the RTWDOG all status flags.*
- static void **RTWDOG\_EnableWindowMode** (RTWDOG\_Type \*base, bool enable)  
*Enables/disables the window mode.*
- static uint32\_t **RTWDOG\_CountToMesec** (RTWDOG\_Type \*base, uint32\_t count, uint32\_t clockFreqInHz)  
*Converts raw count value to millisecond.*
- void **RTWDOG\_ClearStatusFlags** (RTWDOG\_Type \*base, uint32\_t mask)  
*Clears the RTWDOG flag.*
- static void **RTWDOG\_SetTimeoutValue** (RTWDOG\_Type \*base, uint16\_t timeoutCount)

- static void [RTWDOG\\_SetWindowValue](#) (RTWDOG\_Type \*base, uint16\_t windowValue)
 

*Sets the RTWDOG timeout value.*
- [\\_\\_STATIC\\_FORCEINLINE](#) void [RTWDOG\\_Unlock](#) (RTWDOG\_Type \*base)
 

*Unlocks the RTWDOG register written.*
- static void [RTWDOG\\_Refresh](#) (RTWDOG\_Type \*base)
 

*Refreshes the RTWDOG timer.*
- static uint16\_t [RTWDOG\\_GetCounterValue](#) (RTWDOG\_Type \*base)
 

*Gets the RTWDOG counter value.*

## 34.3 Data Structure Documentation

### 34.3.1 struct rtwdog\_work\_mode\_t

#### Data Fields

- bool [enableWait](#)

*Enables or disables RTWDOG in wait mode.*
- bool [enableStop](#)

*Enables or disables RTWDOG in stop mode.*
- bool [enableDebug](#)

*Enables or disables RTWDOG in debug mode.*

### 34.3.2 struct rtwdog\_config\_t

#### Data Fields

- bool [enableRtwdog](#)

*Enables or disables RTWDOG.*
- [rtwdog\\_clock\\_source\\_t](#) [clockSource](#)

*Clock source select.*
- [rtwdog\\_clock\\_prescaler\\_t](#) [prescaler](#)

*Clock prescaler value.*
- [rtwdog\\_work\\_mode\\_t](#) [workMode](#)

*Configures RTWDOG work mode in debug stop and wait mode.*
- [rtwdog\\_test\\_mode\\_t](#) [testMode](#)

*Configures RTWDOG test mode.*
- bool [enableUpdate](#)

*Update write-once register enable.*
- bool [enableInterrupt](#)

*Enables or disables RTWDOG interrupt.*
- bool [enableWindowMode](#)

*Enables or disables RTWDOG window mode.*
- uint16\_t [windowValue](#)

*Window value.*
- uint16\_t [timeoutValue](#)

*Timeout value.*

## 34.4 Macro Definition Documentation

34.4.1 `#define FSL_RTWDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))`

## 34.5 Enumeration Type Documentation

### 34.5.1 enum rtwdog\_clock\_source\_t

Enumerator

*kRTWDOG\_ClockSource0* Clock source 0.

*kRTWDOG\_ClockSource1* Clock source 1.

*kRTWDOG\_ClockSource2* Clock source 2.

*kRTWDOG\_ClockSource3* Clock source 3.

### 34.5.2 enum rtwdog\_clock\_prescaler\_t

Enumerator

*kRTWDOG\_ClockPrescalerDivide1* Divided by 1.

*kRTWDOG\_ClockPrescalerDivide256* Divided by 256.

### 34.5.3 enum rtwdog\_test\_mode\_t

Enumerator

*kRTWDOG\_TestModeDisabled* Test Mode disabled.

*kRTWDOG\_UserModeEnabled* User Mode enabled.

*kRTWDOG\_LowByteTest* Test Mode enabled, only low byte is used.

*kRTWDOG\_HighByteTest* Test Mode enabled, only high byte is used.

### 34.5.4 enum \_rtwdog\_interrupt\_enable\_t

This structure contains the settings for all of the RTWDOG interrupt configurations.

Enumerator

*kRTWDOG\_InterruptEnable* Interrupt is generated before forcing a reset.

### 34.5.5 enum \_rtwdog\_status\_flags\_t

This structure contains the RTWDOG status flags for use in the RTWDOG functions.

Enumerator

- kRTWDOG\_RunningFlag* Running flag, set when RTWDOG is enabled.
- kRTWDOG\_InterruptFlag* Interrupt flag, set when interrupt occurs.

## 34.6 Function Documentation

### 34.6.1 void RTWDOG\_GetDefaultConfig ( rtwdog\_config\_t \* config )

This function initializes the RTWDOG configuration structure to default values. The default values are:

```
* rtwdogConfig->enableRtwdog = true;
* rtwdogConfig->clockSource = kRTWDOG_ClockSource1;
* rtwdogConfig->prescaler = kRTWDOG_ClockPrescalerDivide1;
* rtwdogConfig->workMode.enableWait = true;
* rtwdogConfig->workMode.enableStop = false;
* rtwdogConfig->workMode.enableDebug = false;
* rtwdogConfig->testMode = kRTWDOG_TestModeDisabled;
* rtwdogConfig->enableUpdate = true;
* rtwdogConfig->enableInterrupt = false;
* rtwdogConfig->enableWindowMode = false;
* rtwdogConfig->>windowValue = 0U;
* rtwdogConfig->timeoutValue = 0xFFFFU;
*
```

Parameters

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

See Also

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

### 34.6.2 void RTWDOG\_Init ( RTWDOG\_Type \* base, const rtwdog\_config\_t \* config )

This function initializes the RTWDOG. To reconfigure the RTWDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
* rtwdog_config_t config;
* RTWDOG_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* config.enableUpdate = true;
* RTWDOG_Init(wdog_base,&config);
*
```

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | RTWDOG peripheral base address.  |
| <i>config</i> | The configuration of the RTWDOG. |

### 34.6.3 void RTWDOG\_Deinit ( RTWDOG\_Type \* *base* )

This function shuts down the RTWDOG. Ensure that the WDOG\_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

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

### 34.6.4 static void RTWDOG\_Enable ( RTWDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to enable the RTWDOG. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

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

### 34.6.5 static void RTWDOG\_Disable ( RTWDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to disable the RTWDOG. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

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

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

This function writes a value into the WDOG\_CS register to enable the RTWDOG interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                            |
| <i>mask</i> | The interrupts to enable. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kRTWDOG_InterruptEnable</li></ul> |

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

This function writes a value into the WDOG\_CS register to disable the RTWDOG interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                              |
| <i>mask</i> | The interrupts to disabled. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kRTWDOG_InterruptEnable</li></ul> |

### 34.6.8 static uint32\_t RTWDOG\_GetStatusFlags ( RTWDOG\_Type \* *base* ) [inline], [static]

This function gets all status flags.

Example to get the running flag:

```
* uint32_t status;
* status = RTWDOG_GetStatusFlags(wdog_base) &
* kRTWDOG_RunningFlag;
*
```

Parameters

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

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

[\\_rtwdog\\_status\\_flags\\_t](#)

- true: related status flag has been set.
- false: related status flag is not set.

#### **34.6.9 static void RTWDOG\_EnableWindowMode ( RTWDOG\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | RTWDOG peripheral base address.               |
| <i>enable</i> | Enables(true) or disables(false) the feature. |

#### **34.6.10 static uint32\_t RTWDOG\_CountToMesec ( RTWDOG\_Type \* *base*, uint32\_t *count*, uint32\_t *clockFreqInHz* ) [inline], [static]**

Note that if the clock frequency is too high the timeout period can be less than 1 ms. In this case this api will return 0 value.

Parameters

|                      |                                                |
|----------------------|------------------------------------------------|
| <i>base</i>          | RTWDOG peripheral base address.                |
| <i>count</i>         | Raw count value.                               |
| <i>clockFreqInHz</i> | The frequency of the clock source RTWDOG uses. |

#### **34.6.11 void RTWDOG\_ClearStatusFlags ( RTWDOG\_Type \* *base*, uint32\_t *mask* )**

This function clears the RTWDOG status flag.

Example to clear an interrupt flag:

```
* RTWDOG_ClearStatusFlags(wdog_base,
 kRTWDOG_InterruptFlag);
*
```

## Parameters

|             |                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTWDOG peripheral base address.                                                                                                                                  |
| <i>mask</i> | The status flags to clear. The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• kRTWDOG_InterruptFlag</li></ul> |

**34.6.12 static void RTWDOG\_SetTimeoutValue ( RTWDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]**

This function writes a timeout value into the WDOG\_TOVAL register. The WDOG\_TOVAL register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

## Parameters

|                     |                                                    |
|---------------------|----------------------------------------------------|
| <i>base</i>         | RTWDOG peripheral base address                     |
| <i>timeoutCount</i> | RTWDOG timeout value, count of RTWDOG clock ticks. |

**34.6.13 static void RTWDOG\_SetWindowValue ( RTWDOG\_Type \* *base*, uint16\_t *windowValue* ) [inline], [static]**

This function writes a window value into the WDOG\_WIN register. The WDOG\_WIN register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

## Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | RTWDOG peripheral base address. |
| <i>windowValue</i> | RTWDOG window value.            |

**34.6.14 \_\_STATIC\_FORCEINLINE void RTWDOG\_Unlock ( RTWDOG\_Type \* *base* )**

This function unlocks the RTWDOG register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

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

#### **34.6.15 static void RTWDOG\_Refresh ( RTWDOG\_Type \* *base* ) [inline], [static]**

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

Parameters

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

#### **34.6.16 static uint16\_t RTWDOG\_GetCounterValue ( RTWDOG\_Type \* *base* ) [inline], [static]**

This function gets the RTWDOG counter value.

Parameters

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

Returns

Current RTWDOG counter value.

## **Chapter 35**

### **S3MU: Sentinel**

The firmware lib provides Sentinel driver.

# Chapter 36

## SAI: Serial Audio Interface

### 36.1 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.

### 36.2 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`

### 36.3 Typical use case

#### 36.3.1 SAI Send/receive using an interrupt method

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

#### 36.3.2 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 EDMA Driver](#)

## 36.4 SAI Driver

### 36.4.1 Overview

#### Data Structures

- struct [sai\\_config\\_t](#)  
*SAI user configuration structure. [More...](#)*
- struct [sai\\_transfer\\_format\\_t](#)  
*sai transfer format [More...](#)*
- struct [sai\\_fifo\\_t](#)  
*sai fifo configurations [More...](#)*
- struct [sai\\_bit\\_clock\\_t](#)  
*sai bit clock configurations [More...](#)*
- struct [sai\\_frame\\_sync\\_t](#)  
*sai frame sync configurations [More...](#)*
- struct [sai\\_serial\\_data\\_t](#)  
*sai serial data configurations [More...](#)*
- struct [sai\\_transceiver\\_t](#)  
*sai transceiver configurations [More...](#)*
- struct [sai\\_transfer\\_t](#)  
*SAI transfer structure. [More...](#)*
- struct [sai\\_handle\\_t](#)  
*SAI handle structure. [More...](#)*

#### Macros

- #define [SAI\\_XFER\\_QUEUE\\_SIZE](#) (4U)  
*SAI transfer queue size, user can refine it according to use case.*
- #define [FSL\\_SAI\\_HAS\\_FIFO\\_EXTEND\\_FEATURE](#) 1  
*sai fifo feature*

#### Typedefs

- typedef void(\* [sai\\_transfer\\_callback\\_t](#) )(I2S\_Type \*base, sai\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*SAI transfer callback prototype.*

## Enumerations

- enum {
   
kStatus\_SAI\_TxBusy = MAKE\_STATUS(kStatusGroup\_SAI, 0),
   
kStatus\_SAI\_RxBusy = MAKE\_STATUS(kStatusGroup\_SAI, 1),
   
kStatus\_SAI\_TxError = MAKE\_STATUS(kStatusGroup\_SAI, 2),
   
kStatus\_SAI\_RxError = MAKE\_STATUS(kStatusGroup\_SAI, 3),
   
kStatus\_SAI\_QueueFull = MAKE\_STATUS(kStatusGroup\_SAI, 4),
   
kStatus\_SAI\_TxIdle = MAKE\_STATUS(kStatusGroup\_SAI, 5),
   
kStatus\_SAI\_RxIdle = MAKE\_STATUS(kStatusGroup\_SAI, 6) }
   
*\_sai\_status\_t, SAI return status.*
- enum {
   
kSAI\_Channel0Mask = 1 << 0U,
   
kSAI\_Channel1Mask = 1 << 1U,
   
kSAI\_Channel2Mask = 1 << 2U,
   
kSAI\_Channel3Mask = 1 << 3U,
   
kSAI\_Channel4Mask = 1 << 4U,
   
kSAI\_Channel5Mask = 1 << 5U,
   
kSAI\_Channel6Mask = 1 << 6U,
   
kSAI\_Channel7Mask = 1 << 7U }  
*\_sai\_channel\_mask,.sai channel mask value, actual channel numbers is depend soc specific*
- enum **sai\_protocol\_t** {
   
kSAI\_BusLeftJustified = 0x0U,
   
kSAI\_BusRightJustified,
   
kSAI\_BusI2S,
   
kSAI\_BusPCMA,
   
kSAI\_BusPCMB }
- Define the SAI bus type.*
- enum **sai\_master\_slave\_t** {
   
kSAI\_Master = 0x0U,
   
kSAI\_Slave = 0x1U,
   
kSAI\_Bclk\_Master\_FrameSync\_Slave = 0x2U,
   
kSAI\_Bclk\_Slave\_FrameSync\_Master = 0x3U }  
*Master or slave mode.*
- enum **sai\_mono\_stereo\_t** {
   
kSAI\_Stereo = 0x0U,
   
kSAI\_MonoRight,
   
kSAI\_MonoLeft }
- Mono or stereo audio format.*
- enum **sai\_data\_order\_t** {
   
kSAI\_DataLSB = 0x0U,
   
kSAI\_DataMSB }
- SAI data order, MSB or LSB.*
- enum **sai\_clock\_polarity\_t** {

- ```
kSAI_PolarityActiveHigh = 0x0U,
kSAI_PolarityActiveLow = 0x1U,
kSAI_SampleOnFallingEdge = 0x0U,
kSAI_SampleOnRisingEdge = 0x1U }
```

SAI clock polarity, active high or low.
- enum `sai_sync_mode_t` {


```
kSAI_ModeAsync = 0x0U,
kSAI_ModeSync }
```

Synchronous or asynchronous mode.
- enum `sai_bclk_source_t` {


```
kSAI_BclkSourceBusclk = 0x0U,
kSAI_BclkSourceMclkOption1 = 0x1U,
kSAI_BclkSourceMclkOption2 = 0x2U,
kSAI_BclkSourceMclkOption3 = 0x3U,
kSAI_BclkSourceMclkDiv = 0x1U,
kSAI_BclkSourceOtherSai0 = 0x2U,
kSAI_BclkSourceOtherSai1 = 0x3U }
```

Bit clock source.
- enum {


```
kSAI_WordStartInterruptEnable,
kSAI_SyncErrorInterruptEnable = I2S_TCSR_SEIE_MASK,
kSAI_FIFOWarningInterruptEnable = I2S_TCSR_FWIE_MASK,
kSAI_FIFOErrorInterruptEnable = I2S_TCSR_FEIE_MASK,
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_t` {


```
kSAI_ResetTypeSoftware = I2S_TCSR_SR_MASK,
kSAI_ResetTypeFIFO = I2S_TCSR_FR_MASK,
kSAI_ResetAll = I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK }
```

The reset type.
- enum `sai_fifo_packing_t` {


```
kSAI_FifoPackingDisabled = 0x0U,
kSAI_FifoPacking8bit = 0x2U,
kSAI_FifoPacking16bit = 0x3U }
```

The SAI packing mode The mode includes 8 bit and 16 bit packing.
- enum `sai_sample_rate_t` {

```
kSAI_SampleRate8KHz = 8000U,
kSAI_SampleRate11025Hz = 11025U,
kSAI_SampleRate12KHz = 12000U,
kSAI_SampleRate16KHz = 16000U,
kSAI_SampleRate22050Hz = 22050U,
kSAI_SampleRate24KHz = 24000U,
kSAI_SampleRate32KHz = 32000U,
kSAI_SampleRate44100Hz = 44100U,
kSAI_SampleRate48KHz = 48000U,
kSAI_SampleRate96KHz = 96000U,
kSAI_SampleRate192KHz = 192000U,
kSAI_SampleRate384KHz = 384000U }
```

Audio sample rate.

- enum `sai_word_width_t` {

kSAI_WordWidth8bits = 8U,

kSAI_WordWidth16bits = 16U,

kSAI_WordWidth24bits = 24U,

kSAI_WordWidth32bits = 32U }

Audio word width.

- enum `sai_data_pin_state_t` {

kSAI_DataPinStateTriState,

kSAI_DataPinStateOutputZero = 1U }
- sai data pin state definition*
- enum `sai_fifo_combine_t` {

kSAI_FifoCombineDisabled = 0U,

kSAI_FifoCombineModeEnabledOnRead,

kSAI_FifoCombineModeEnabledOnWrite,

kSAI_FifoCombineModeEnabledReadWrite }
- sai fifo combine mode definition*

- enum `sai_transceiver_type_t` {

kSAI_Transmitter = 0U,

kSAI_Receiver = 1U }
- sai transceiver type*
- enum `sai_frame_sync_len_t` {

kSAI_FrameSyncLenOneBitClk = 0U,

kSAI_FrameSyncLenPerWordWidth = 1U }
- sai frame sync len*

Driver version

- #define `FSL_SAI_DRIVER_VERSION` (`MAKE_VERSION(2, 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_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_TxSetFIFO Packing** (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)

Tx interrupt handler.

36.4.2 Data Structure Documentation

36.4.2.1 struct sai_config_t

Data Fields

- [sai_protocol_t](#) protocol

Audio bus protocol in SAI.
- [sai_sync_mode_t](#) syncMode

SAI sync mode, control Tx/Rx clock sync.

- `sai_bclk_source_t bclkSource`
Bit Clock source.
- `sai_master_slave_t masterSlave`
Master or slave.

36.4.2.2 struct sai_transfer_format_t

Data Fields

- `uint32_t sampleRate_Hz`
Sample rate of audio data.
- `uint32_t bitWidth`
Data length of audio data, usually 8/16/24/32 bits.
- `sai_mono_stereo_t stereo`
Mono or stereo.
- `uint8_t 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_t::isFrameSyncCompact`

36.4.2.3 struct sai_fifo_t

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

36.4.2.4 struct sai_bit_clock_t

Data Fields

- bool `bclkSrcSwap`
bit clock source swap
- bool `bclkInputDelay`
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .
- `sai_clock_polarity_t bclkPolarity`
bit clock polarity
- `sai_bclk_source_t bclkSource`
bit Clock source

Field Documentation

(1) `bool sai_bit_clock_t::bclkInputDelay`

36.4.2.5 struct sai_frame_sync_t

Data Fields

- `uint8_t frameSyncWidth`
frame sync width in number of bit clocks
- bool `frameSyncEarly`
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.
- bool `frameSyncGenerateOnDemand`
internal frame sync is generated when FIFO waring flag is clear
- `sai_clock_polarity_t frameSyncPolarity`
frame sync polarity

36.4.2.6 struct sai_serial_data_t

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

36.4.2.7 struct sai_transceiver_t

Data Fields

- `sai_serial_data_t serialData`
serial data configurations
- `sai_frame_sync_t frameSync`
ws configurations
- `sai_bit_clock_t bitClock`
bit clock configurations
- `sai_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.

36.4.2.8 struct sai_transfer_t

Data Fields

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

Field Documentation

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

36.4.2.9 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, reference _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.

36.4.3 Macro Definition Documentation

36.4.3.1 #define SAI_XFER_QUEUE_SIZE (4U)

36.4.4 Enumeration Type Documentation

36.4.4.1 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.

36.4.4.2 anonymous enum

Enumerator

<i>kSAI_Channel0Mask</i>	channel 0 mask value
<i>kSAI_Channel1Mask</i>	channel 1 mask value
<i>kSAI_Channel2Mask</i>	channel 2 mask value
<i>kSAI_Channel3Mask</i>	channel 3 mask value
<i>kSAI_Channel4Mask</i>	channel 4 mask value
<i>kSAI_Channel5Mask</i>	channel 5 mask value
<i>kSAI_Channel6Mask</i>	channel 6 mask value
<i>kSAI_Channel7Mask</i>	channel 7 mask value

36.4.4.3 enum sai_protocol_t

Enumerator

<i>kSAI_BusLeftJustified</i>	Uses left justified format.
<i>kSAI_BusRightJustified</i>	Uses right justified format.
<i>kSAI_BusI2S</i>	Uses I2S format.
<i>kSAI_BusPCMA</i>	Uses I2S PCM A format.
<i>kSAI_BusPCMB</i>	Uses I2S PCM B format.

36.4.4.4 enum sai_master_slave_t

Enumerator

<i>kSAI_Master</i>	Master mode include bclk and frame sync.
<i>kSAI_Slave</i>	Slave mode include bclk and frame sync.
<i>kSAI_Bclk_Master_FrameSync_Slave</i>	bclk in master mode, frame sync in slave mode
<i>kSAI_Bclk_Slave_FrameSync_Master</i>	bclk in slave mode, frame sync in master mode

36.4.4.5 enum sai_mono_stereo_t

Enumerator

<i>kSAI_Stereo</i>	Stereo sound.
<i>kSAI_MonoRight</i>	Only Right channel have sound.
<i>kSAI_MonoLeft</i>	Only left channel have sound.

36.4.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB LSB bit transferred first.

kSAI_DataMSB MSB bit transferred first.

36.4.4.7 enum sai_clock_polarity_t

Enumerator

kSAI_PolarityActiveHigh Drive outputs on rising edge.

kSAI_PolarityActiveLow Drive outputs on falling edge.

kSAI_SampleOnFallingEdge Sample inputs on falling edge.

kSAI_SampleOnRisingEdge Sample inputs on rising edge.

36.4.4.8 enum sai_sync_mode_t

Enumerator

kSAI_ModeAsync Asynchronous mode.

kSAI_ModeSync Synchronous mode (with receiver or transmit)

36.4.4.9 enum sai_bclk_source_t

Enumerator

kSAI_BclkSourceBusclk Bit clock using bus clock.

kSAI_BclkSourceMclkOption1 Bit clock MCLK option 1.

kSAI_BclkSourceMclkOption2 Bit clock MCLK option2.

kSAI_BclkSourceMclkOption3 Bit clock MCLK option3.

kSAI_BclkSourceMclkDiv Bit clock using master clock divider.

kSAI_BclkSourceOtherSai0 Bit clock from other SAI device.

kSAI_BclkSourceOtherSai1 Bit clock from other SAI device.

36.4.4.10 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.

36.4.4.11 anonymous enum

Enumerator

kSAI_FIFOWarningDMAEnable FIFO warning caused by the DMA request.

kSAI_FIFORequestDMAEnable FIFO request caused by the DMA request.

36.4.4.12 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.

36.4.4.13 enum sai_reset_type_t

Enumerator

kSAI_ResetTypeSoftware Software reset, reset the logic state.

kSAI_ResetTypeFIFO FIFO reset, reset the FIFO read and write pointer.

kSAI_ResetAll All reset.

36.4.4.14 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.

kSAI_FifoPacking8bit 8 bit packing enabled

kSAI_FifoPacking16bit 16bit packing enabled

36.4.4.15 enum sai_sample_rate_t

Enumerator

kSAI_SampleRate8KHz Sample rate 8000 Hz.

kSAI_SampleRate11025Hz Sample rate 11025 Hz.

kSAI_SampleRate12KHz Sample rate 12000 Hz.

kSAI_SampleRate16KHz Sample rate 16000 Hz.

kSAI_SampleRate22050Hz Sample rate 22050 Hz.

kSAI_SampleRate24KHz Sample rate 24000 Hz.

kSAI_SampleRate32KHz Sample rate 32000 Hz.

kSAI_SampleRate44100Hz Sample rate 44100 Hz.
kSAI_SampleRate48KHz Sample rate 48000 Hz.
kSAI_SampleRate96KHz Sample rate 96000 Hz.
kSAI_SampleRate192KHz Sample rate 192000 Hz.
kSAI_SampleRate384KHz Sample rate 384000 Hz.

36.4.4.16 enum sai_word_width_t

Enumerator

kSAI_WordWidth8bits Audio data width 8 bits.
kSAI_WordWidth16bits Audio data width 16 bits.
kSAI_WordWidth24bits Audio data width 24 bits.
kSAI_WordWidth32bits Audio data width 32 bits.

36.4.4.17 enum sai_data_pin_state_t

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

36.4.4.18 enum sai_fifo_combine_t

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

36.4.4.19 enum sai_transceiver_type_t

Enumerator

kSAI_Transmitter sai transmitter
kSAI_Receiver sai receiver

36.4.4.20 enum sai_frame_sync_len_t

Enumerator

kSAI_FrameSyncLenOneBitClk 1 bit clock frame sync len for DSP mode

kSAI_FrameSyncLenPerWordWidth Frame sync length decided by word width.

36.4.5 Function Documentation

36.4.5.1 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.
-------------	-------------------

36.4.5.2 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.
-------------	-------------------

36.4.5.3 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
-------------	------------------

36.4.5.4 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
-------------	------------------

36.4.5.5 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.

36.4.5.6 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.

36.4.5.7 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.

36.4.5.8 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.

36.4.5.9 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.

36.4.5.10 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.

36.4.5.11 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.

36.4.5.12 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.

36.4.5.13 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.

36.4.5.14 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.

36.4.5.15 void SAI_TxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

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

<i>config</i>	fifo configurations.
---------------	----------------------

36.4.5.16 void SAI_RxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

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

36.4.5.17 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.

36.4.5.18 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.

36.4.5.19 void SAI_TxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

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

36.4.5.20 void SAI_RxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

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

36.4.5.21 void SAI_TxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

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

36.4.5.22 void SAI_RxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

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

36.4.5.23 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- Mask</i>	mask value of the channel to be enable.

36.4.5.24 void SAI_GetLeftJustifiedConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

**36.4.5.25 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.

36.4.5.26 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.

36.4.5.27 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.

36.4.5.28 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.

36.4.5.29 static void SAI_TxClearStatusFlags (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 source if defined:</p> <ul style="list-style-type: none"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

36.4.5.30 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.

36.4.5.31 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"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

36.4.5.32 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

36.4.5.33 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

36.4.5.34 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.

36.4.5.35 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.

36.4.5.36 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

36.4.5.37 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

36.4.5.38 void SAI_TxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

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

36.4.5.39 void SAI_RxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

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

36.4.5.40 void SAI_TxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

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

36.4.5.41 void SAI_RxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

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

36.4.5.42 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.

36.4.5.43 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.

36.4.5.44 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.

**36.4.5.45 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.

36.4.5.46 static void SAI_TxEnableInterrupts (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"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.47 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"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.48 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"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.49 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"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.50 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"> • kSAI_FIFOWarningDMAEnable • kSAI_FIFORequestDMAEnable

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

36.4.5.51 static void SAI_RxEnableDMA (I2S_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

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

36.4.5.52 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.

36.4.5.53 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.

36.4.5.54 void SAI_WriteBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

36.4.5.55 void SAI_WriteMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

**36.4.5.56 static void SAI_WriteData (I2S_Type * *base*, uint32_t *channel*, uint32_t *data*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>data</i>	Data needs to be written.

36.4.5.57 void SAI_ReadBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

36.4.5.58 void SAI_ReadMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

```
36.4.5.59 static uint32_t SAI_ReadData ( I2S_Type * base, uint32_t channel )  
[inline], [static]
```

Parameters

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

Returns

Data in SAI FIFO.

36.4.5.60 void SAI_TransferTxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function

36.4.5.61 void SAI_TransferRxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function.

36.4.5.62 void SAI_TransferTxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	transmitter configurations.

36.4.5.63 void SAI_TransferRxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	receiver configurations.

36.4.5.64 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.

36.4.5.65 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.

36.4.5.66 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.

36.4.5.67 **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.

36.4.5.68 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.

36.4.5.69 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.

36.4.5.70 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.

36.4.5.71 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.

36.4.5.72 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.

36.4.5.73 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.

36.5 SAI EDMA Driver

36.5.1 Overview

Data Structures

- struct `sai_edma_handle_t`
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef void(* `sai_edma_callback_t`)(I2S_Type *base, sai_edma_handle_t *handle, `status_t` status, void *userData)
SAI eDMA transfer callback function for finish and error.

Enumerations

- enum `sai_edma_interleave_t` {

`kSAI_EDMAInterleavePerChannelSample`,

`kSAI_EDMAInterleavePerChannelBlock` }
- sai interleave type*

Driver version

- #define `FSL_SAI_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 7, 0)`)
Version 2.7.0.

eDMA Transactional

- void `SAI_TransferTxCreateHandleEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_edma_callback_t` callback, void *userData, `edma_handle_t` *txDmaHandle)
Initializes the SAI eDMA handle.
- void `SAI_TransferRxCreateHandleEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_edma_callback_t` callback, void *userData, `edma_handle_t` *rxDmaHandle)
Initializes the SAI Rx eDMA handle.
- void `SAI_TransferSetInterleaveType` (sai_edma_handle_t *handle, `sai_edma_interleave_t` interleaveType)
Initializes the SAI interleave type.
- void `SAI_TransferTxSetConfigEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transceiver_t` *saiConfig)
Configures the SAI Tx.
- void `SAI_TransferRxSetConfigEDMA` (I2S_Type *base, sai_edma_handle_t *handle, `sai_transceiver_t` *saiConfig)
Configures the SAI Rx.

- `status_t SAI_TransferSendEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer)`
Performs a non-blocking SAI transfer using eDMA.
- `status_t SAI_TransferReceiveEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer)`
Performs a non-blocking SAI receive using eDMA.
- `status_t SAI_TransferSendLoopEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)`
Performs a non-blocking SAI loop transfer using eDMA.
- `status_t SAI_TransferReceiveLoopEDMA (I2S_Type *base, sai_edma_handle_t *handle, sai_transfer_t *xfer, uint32_t loopTransferCount)`
Performs a non-blocking SAI loop transfer using eDMA.
- `void SAI_TransferTerminateSendEDMA (I2S_Type *base, sai_edma_handle_t *handle)`
Terminate all SAI send.
- `void SAI_TransferTerminateReceiveEDMA (I2S_Type *base, sai_edma_handle_t *handle)`
Terminate all SAI receive.
- `void SAI_TransferAbortSendEDMA (I2S_Type *base, sai_edma_handle_t *handle)`
Aborts a SAI transfer using eDMA.
- `void SAI_TransferAbortReceiveEDMA (I2S_Type *base, sai_edma_handle_t *handle)`
Aborts a SAI receive using eDMA.
- `status_t SAI_TransferGetSendCountEDMA (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)`
Gets byte count sent by SAI.
- `status_t SAI_TransferGetReceiveCountEDMA (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)`
Gets byte count received by SAI.
- `uint32_t SAI_TransferGetValidTransferSlotsEDMA (I2S_Type *base, sai_edma_handle_t *handle)`
Gets valid transfer slot.

36.5.2 Data Structure Documentation

36.5.2.1 struct sai_edma_handle

Data Fields

- `edma_handle_t * dmaHandle`
DMA handler for SAI send.
- `uint8_t nbytes`
eDMA minor byte transfer count initially configured.
- `uint8_t bytesPerFrame`
Bytes in a frame.
- `uint8_t channelMask`
Enabled channel mask value, reference _sai_channel_mask.
- `uint8_t channelNums`
total enabled channel nums
- `uint8_t channel`
Which data channel.
- `uint8_t count`

- `uint32_t state`
The transfer data count in a DMA request.
- `sai_edma_callback_t callback`
Internal state for SAI eDMA transfer.
- `void *userData`
Callback for users while transfer finish or error occurs.
- `uint8_t tcd [(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`
TCD pool for eDMA transfer.
- `sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]`
Transfer queue storing queued transfer.
- `size_t transferSize [SAI_XFER_QUEUE_SIZE]`
Data bytes need to transfer.
- `sai_edma_interleave_t interleaveType`
Transfer interleave type.
- `volatile uint8_t queueUser`
Index for user to queue transfer.
- `volatile uint8_t queueDriver`
Index for driver to get the transfer data and size.

Field Documentation

- (1) `uint8_t sai_edma_handle_t::nbytes`
- (2) `uint8_t sai_edma_handle_t::tcd[(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`
- (3) `sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`
- (4) `volatile uint8_t sai_edma_handle_t::queueUser`

36.5.3 Enumeration Type Documentation

36.5.3.1 enum sai_edma_interleave_t

SAI data interleave per channel sample

|LEFT CHANNEL | RIGHT CHANNEL | LEFT CHANNEL | RIGHT CHANNEL | LEFT CHANNEL | RIGHT CHANNEL | |

SAI data interleave per channel block

|LEFT CHANNEL | LEFT CHANNEL | LEFT CHANNEL | ... | **RIGHT CHANNEL | RIGHT CHANNEL | RIGHT CHANNEL | ...|**

36.5.4 Function Documentation

36.5.4.1 `void SAI_TransferTxCreateHandleEDMA (I2S_Type * base, sai_edma_handle_t * handle, sai_edma_callback_t callback, void * userData, edma_handle_t * txDmaHandle)`

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

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>base</i>	SAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>txDmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

36.5.4.2 void SAI_TransferRxCreateHandleEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_edma_callback_t *callback*, void * *userData*, edma_handle_t * *rxDmaHandle*)

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

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>base</i>	SAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>rxDmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

36.5.4.3 void SAI_TransferSetInterleaveType (sai_edma_handle_t * *handle*, sai_edma_interleave_t *interleaveType*)

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI_EDMAInterleavePerChannelSample always

Parameters

<i>handle</i>	SAI eDMA handle pointer.
---------------	--------------------------

<i>interleaveType</i>	Multi channel interleave type.
-----------------------	--------------------------------

36.5.4.4 void SAI_TransferTxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Enumerator

Note

kSAI_EDMAInterleavePerChannelSample **kSAI_EDMAInterleavePerChannelBlock** SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of *_sai_channel_mask* enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnWrite to the fifoCombine member of *sai_fifo_combine_t* which is a member of *sai_transceiver_t*. This is an example of multi-channel data transfer configuration step.

```
*     sai_transceiver_t config;
*     SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits,
*                             kSAI_Stereo, kSAI_Channel0Mask|kSAI_Channel1Mask);
*     config fifo fifoCombine = kSAI_FifoCombineModeEnabledOnWrite
*     ;
*     SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
*
```

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

36.5.4.5 void SAI_TransferRxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Note

SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of *sai_transceiver_t* with the corresponding values of *_sai_channel_mask* enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of *sai_fifo_combine_t* which is a member of *sai_transceiver_t*. This is an example of multi-channel data transfer configuration step.

```
*     sai_transceiver_t config;
*     SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits,
*                             kSAI_Stereo, kSAI_Channel0Mask|kSAI_Channel1Mask);
*     config fifo fifoCombine = kSAI_FifoCombineModeEnabledOnRead
*     ;
*     SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
*
```

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

36.5.4.6 status_t SAI_TransferSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

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

This function support multi channel transfer,

1. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
2. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_TxBusy</i>	SAI is busy sending data.

36.5.4.7 status_t SAI_TransferReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

This function support multi channel transfer,

1. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
2. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

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

36.5.4.8 **status_t SAI_TransferSendLoopEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*, uint32_t *loopTransferCount*)**

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).

<i>loopTransferCount</i>	the counts of xfer array.
--------------------------	---------------------------

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

36.5.4.9 status_t SAI_TransferReceiveLoopEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*, uint32_t *loopTransferCount*)

Note

This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure, should be a array with elements counts >=1(<i>loopTransferCount</i>).
<i>loopTransferCount</i>	the counts of xfer array.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

36.5.4.10 void SAI_TransferTerminateSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

36.5.4.11 void SAI_TransferTerminateReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

36.5.4.12 void SAI_TransferAbortSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

36.5.4.13 void SAI_TransferAbortReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

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

<i>handle</i>	SAI eDMA handle pointer.
---------------	--------------------------

36.5.4.14 status_t SAI_TransferGetSendCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count sent by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

36.5.4.15 status_t SAI_TransferGetReceiveCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count received by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

36.5.4.16 uint32_t SAI_TransferGetValidTransferSlotsEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.

Return values

<i>valid</i>	slot count that application submit.
--------------	-------------------------------------

Chapter 37

SEMC: Smart External DRAM Controller Driver

37.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Smart External DRAM Controller block of MCUXpresso SDK devices.

37.2 SEMC: Smart External DRAM Controller Driver

37.2.1 SEMC Initialization Operation

The SEMC Initialize is to initialize for common configure: gate the SEMC clock, configure IOMUX, and queue weight setting. The SEMC Deinitialize is to ungate the clock and disable SEMC module.

37.2.2 SEMC Interrupt Operation

The interrupt and disable operation for SEMC.

37.2.3 SEMC Memory access Operation

This group is mainly provide NAND/NOR memory access API which is through IP bus/ IP command access. Since the AXI access is directly read/write is so easy, so the AXI read/write part is not provided in SEMC.

37.3 Typical use case

Data Structures

- struct `semc_sdram_config_t`
SEMC SDRAM configuration structure. [More...](#)
- struct `semc_nand_timing_config_t`
SEMC NAND device timing configuration structure. [More...](#)
- struct `semc_nand_config_t`
SEMC NAND configuration structure. [More...](#)
- struct `semc_nor_config_t`
SEMC NOR configuration structure. [More...](#)
- struct `semc_sram_config_t`
SEMC SRAM configuration structure. [More...](#)
- struct `semc_dbi_config_t`
SEMC DBI configuration structure. [More...](#)
- struct `semc_queuea_weight_struct_t`

- union `semc_queuea_weight_t`
SEMC AXI queue a weight setting structure. [More...](#)
- struct `semc_queueb_weight_struct_t`
SEMC AXI queue b weight setting structure. [More...](#)
- union `semc_queueb_weight_t`
SEMC AXI queue b weight setting union. [More...](#)
- struct `semc_axi_queueweight_t`
SEMC AXI queue weight setting. [More...](#)
- struct `semc_config_t`
SEMC configuration structure. [More...](#)

Enumerations

- enum {

`kStatus_SEMC_InvalidDeviceType` = MAKE_STATUS(kStatusGroup_SEMC, 0),

`kStatus_SEMC_IpCommandExecutionError` = MAKE_STATUS(kStatusGroup_SEMC, 1),

`kStatus_SEMC_AxiCommandExecutionError` = MAKE_STATUS(kStatusGroup_SEMC, 2),

`kStatus_SEMC_InvalidMemorySize` = MAKE_STATUS(kStatusGroup_SEMC, 3),

`kStatus_SEMC_InvalidIpCmdDataSize` = MAKE_STATUS(kStatusGroup_SEMC, 4),

`kStatus_SEMC_InvalidAddressPortWidth` = MAKE_STATUS(kStatusGroup_SEMC, 5),

`kStatus_SEMC_InvalidDataPortWidth` = MAKE_STATUS(kStatusGroup_SEMC, 6),

`kStatus_SEMC_InvalidSwPinmuxSelection` = MAKE_STATUS(kStatusGroup_SEMC, 7),

`kStatus_SEMC_InvalidBurstLength` = MAKE_STATUS(kStatusGroup_SEMC, 8),

`kStatus_SEMC_InvalidColumnAddressBitWidth` = MAKE_STATUS(kStatusGroup_SEMC, 9),

`kStatus_SEMC_InvalidBaseAddress` = MAKE_STATUS(kStatusGroup_SEMC, 10),

`kStatus_SEMC_InvalidTimerSetting` = MAKE_STATUS(kStatusGroup_SEMC, 11) }

SEMC status, _semc_status.
- enum `semc_mem_type_t` {

`kSEMC_MemType_SDRAM` = 0,

`kSEMC_MemType_SRAM`,

`kSEMC_MemType_NOR`,

`kSEMC_MemType_NAND`,

`kSEMC_MemType_8080` }

SEMC memory device type.
- enum `semc_waitready_polarity_t` {

`kSEMC_LowActive` = 0,

`kSEMC_HighActive` }

SEMC WAIT/RDY polarity.
- enum `semc_sdram_cs_t` {

`kSEMC_SDRAM_CS0` = 0,

`kSEMC_SDRAM_CS1`,

`kSEMC_SDRAM_CS2`,

`kSEMC_SDRAM_CS3` }

SEMC SDRAM Chip selection .
- enum `semc_sram_cs_t` {

- ```
kSEMC_SRAM_CS0 = 0,
kSEMC_SRAM_CS1,
kSEMC_SRAM_CS2,
kSEMC_SRAM_CS3 }
```

*SEMC SRAM Chip selection.*
- enum `semc_nand_access_type_t` {

```
kSEMC_NAND_ACCESS_BY_AXI = 0,
kSEMC_NAND_ACCESS_BY_IPCMD }
```

*SEMC NAND device type.*
- enum `semc_interrupt_enable_t` {

```
kSEMC_IPCmdDoneInterrupt = SEMC_INTEN_IPCMDDONEEN_MASK,
kSEMC_IPCmdErrInterrupt = SEMC_INTEN_IPCMDERREN_MASK,
kSEMC_AXICmdErrInterrupt = SEMC_INTEN_AXICMDERREN_MASK,
kSEMC_AXIBusErrInterrupt = SEMC_INTEN_AXIBUSERREN_MASK }
```

*SEMC interrupts.*
- enum `semc_ipcmd_datasize_t` {

```
kSEMC_IPcmdDataSize_1bytes = 1,
kSEMC_IPcmdDataSize_2bytes,
kSEMC_IPcmdDataSize_3bytes,
kSEMC_IPcmdDataSize_4bytes }
```

*SEMC IP command data size in bytes.*
- enum `semc_refresh_time_t` {

```
kSEMC_RefreshThreeClocks = 0x0U,
kSEMC_RefreshSixClocks,
kSEMC_RefreshNineClocks }
```

*SEMC auto-refresh timing.*
- enum `semc_caslatency_t` {

```
kSEMC_LatencyOne = 1,
kSEMC_LatencyTwo,
kSEMC_LatencyThree }
```

*CAS latency.*
- enum `semc_sdram_column_bit_num_t` {

```
kSEMC_SdramColumn_12bit = 0x0U,
kSEMC_SdramColumn_11bit,
kSEMC_SdramColumn_10bit,
kSEMC_SdramColumn_9bit,
kSEMC_SdramColumn_8bit }
```

*SEMC sdram column address bit number.*
- enum `sem_sdram_burst_len_t` {

```
kSEMC_Sdram_BurstLen1 = 0,
kSEMC_Sdram_BurstLen2,
kSEMC_Sdram_BurstLen4,
kSEMC_Sdram_BurstLen8 }
```

*SEMC sdram burst length.*
- enum `semc_nand_column_bit_num_t` {

```
kSEMC_NandColum_16bit = 0x0U,
kSEMC_NandColum_15bit,
kSEMC_NandColum_14bit,
kSEMC_NandColum_13bit,
kSEMC_NandColum_12bit,
kSEMC_NandColum_11bit,
kSEMC_NandColum_10bit,
kSEMC_NandColum_9bit }
```

*SEMC nand column address bit number.*

- enum `sem_nand_burst_len_t` {
   
kSEMC\_Nand\_BurstLen1 = 0,
   
kSEMC\_Nand\_BurstLen2,
   
kSEMC\_Nand\_BurstLen4,
   
kSEMC\_Nand\_BurstLen8,
   
kSEMC\_Nand\_BurstLen16,
   
kSEMC\_Nand\_BurstLen32,
   
kSEMC\_Nand\_BurstLen64 }

*SEMC nand burst length.*

- enum `semc_norsram_column_bit_num_t` {
   
kSEMC\_NorColum\_12bit = 0x0U,
   
kSEMC\_NorColum\_11bit,
   
kSEMC\_NorColum\_10bit,
   
kSEMC\_NorColum\_9bit,
   
kSEMC\_NorColum\_8bit,
   
kSEMC\_NorColum\_7bit,
   
kSEMC\_NorColum\_6bit,
   
kSEMC\_NorColum\_5bit,
   
kSEMC\_NorColum\_4bit,
   
kSEMC\_NorColum\_3bit,
   
kSEMC\_NorColum\_2bit }

*SEMC nor/sram column address bit number.*

- enum `sem_norsram_burst_len_t` {
   
kSEMC\_Nor\_BurstLen1 = 0,
   
kSEMC\_Nor\_BurstLen2,
   
kSEMC\_Nor\_BurstLen4,
   
kSEMC\_Nor\_BurstLen8,
   
kSEMC\_Nor\_BurstLen16,
   
kSEMC\_Nor\_BurstLen32,
   
kSEMC\_Nor\_BurstLen64 }

*SEMC nor/sram burst length.*

- enum `semc_dbi_column_bit_num_t` {

```
kSEMC_Dbi_Colum_12bit = 0x0U,
kSEMC_Dbi_Colum_11bit,
kSEMC_Dbi_Colum_10bit,
kSEMC_Dbi_Colum_9bit,
kSEMC_Dbi_Colum_8bit,
kSEMC_Dbi_Colum_7bit,
kSEMC_Dbi_Colum_6bit,
kSEMC_Dbi_Colum_5bit,
kSEMC_Dbi_Colum_4bit,
kSEMC_Dbi_Colum_3bit,
kSEMC_Dbi_Colum_2bit }
```

*SEMC dbi column address bit number.*

- enum sem\_dbi\_burst\_len\_t {
 kSEMC\_Dbi\_BurstLen1 = 0,
 kSEMC\_Dbi\_BurstLen2,
 kSEMC\_Dbi\_Dbi\_BurstLen4,
 kSEMC\_Dbi\_BurstLen8,
 kSEMC\_Dbi\_BurstLen16,
 kSEMC\_Dbi\_BurstLen32,
 kSEMC\_Dbi\_BurstLen64 }

*SEMC dbi burst length.*

- enum semc\_iomux\_pin {
 kSEMC\_MUXA8 = SEMC\_IOCR\_MUX\_A8\_SHIFT,
 kSEMC\_MUXCSX0 = SEMC\_IOCR\_MUX\_CSX0\_SHIFT,
 kSEMC\_MUXCSX1 = SEMC\_IOCR\_MUX\_CSX1\_SHIFT,
 kSEMC\_MUXCSX2 = SEMC\_IOCR\_MUX\_CSX2\_SHIFT,
 kSEMC\_MUXCSX3 = SEMC\_IOCR\_MUX\_CSX3\_SHIFT,
 kSEMC\_MUXRDY = SEMC\_IOCR\_MUX\_RDY\_SHIFT }

*SEMC IOMUXC.*

- enum semc\_iomux\_nora27\_pin {
 kSEMC\_MORA27\_NONE = 0,
 kSEMC\_NORA27\_MUXCSX3 = SEMC\_IOCR\_MUX\_CSX3\_SHIFT,
 kSEMC\_NORA27\_MUXRDY = SEMC\_IOCR\_MUX\_RDY\_SHIFT }

*SEMC NOR/PSRAM Address bit 27 A27.*

- enum smec\_port\_size\_t {
 kSEMC\_PortSize8Bit = 0,
 kSEMC\_PortSize16Bit,
 kSEMC\_PortSize32Bit }

*SEMC port size.*

- enum semc\_addr\_mode\_t {
 kSEMC\_AddrDataMux = 0,
 kSEMC\_AdvAddrdataMux,
 kSEMC\_AddrDataNonMux }

*SEMC address mode.*

- enum semc\_dqs\_mode\_t {
 kSEMC\_Loopbackinternal = 0,

- ```
kSEMC_Loopbackdqspad }
```

SEMC DQS read strobe mode.
- enum `semc_adv_polarity_t` {
 `kSEMC_AdvActiveLow` = 0,
`kSEMC_AdvActiveHigh` }

SEMC ADV signal active polarity.

- enum `semc_sync_mode_t` {
 `kSEMC_AsyncMode` = 0,
`kSEMC_SyncMode` }

SEMC sync mode.

- enum `semc_adv_level_control_t` {
 `kSEMC_AdvHigh` = 0,
`kSEMC_AdvLow` }

SEMC ADV signal level control.

- enum `semc_rdy_polarity_t` {
 `kSEMC_RdyActiveLow` = 0,
`kSEMC_RdyActivehigh` }

SEMC RDY signal active polarity.

- enum `semc_ipcmd_nand_addrmode_t` {
 `kSEMC_NANDAM_ColumnRow` = 0x0U,
`kSEMC_NANDAM_ColumnCA0`,
`kSEMC_NANDAM_ColumnCA0CA1`,
`kSEMC_NANDAM_RawRA0`,
`kSEMC_NANDAM_RawRA0RA1`,
`kSEMC_NANDAM_RawRA0RA1RA2` }

SEMC IP command for NAND: address mode.

- enum `semc_ipcmd_nand_cmdmode_t` {
 `kSEMC_NANDCM_Command` = 0x2U,
`kSEMC_NANDCM_CommandHold`,
`kSEMC_NANDCM_CommandAddress`,
`kSEMC_NANDCM_CommandAddressHold`,
`kSEMC_NANDCM_CommandAddressRead`,
`kSEMC_NANDCM_CommandAddressWrite`,
`kSEMC_NANDCM_CommandRead`,
`kSEMC_NANDCM_CommandWrite`,
`kSEMC_NANDCM_Read`,
`kSEMC_NANDCM_Write` }

SEMC IP command for NAND command mode.

- enum `semc_nand_address_option_t` {
 `kSEMC_NandAddrOption_5byte_CA2RA3` = 0U,
`kSEMC_NandAddrOption_4byte_CA2RA2` = 2U,
`kSEMC_NandAddrOption_3byte_CA2RA1` = 4U,
`kSEMC_NandAddrOption_4byte_CA1RA3` = 1U,
`kSEMC_NandAddrOption_3byte_CA1RA2` = 3U,
`kSEMC_NandAddrOption_2byte_CA1RA1` = 7U }

SEMC NAND address option.

- enum `semc_ipcmd_nor_dbi_t` {

```

kSEMC_NORDBICM_Read = 0x2U,
kSEMC_NORDBICM_Write }

    SEMC IP command for NOR.

• enum semc_ipcmd_sram_t {
    kSEMC_SRAMCM_ArrayRead = 0x2U,
    kSEMC_SRAMCM_ArrayWrite,
    kSEMC_SRAMCM_RegRead,
    kSEMC_SRAMCM_RegWrite }

    SEMC IP command for SRAM.

• enum semc_ipcmd_sdram_t {
    kSEMC_SDRAMCM_Read = 0x8U,
    kSEMC_SDRAMCM_Write,
    kSEMC_SDRAMCM_Modeset,
    kSEMC_SDRAMCM_Active,
    kSEMC_SDRAMCM_AutoRefresh,
    kSEMC_SDRAMCM_SelfRefresh,
    kSEMC_SDRAMCM_Prefetch,
    kSEMC_SDRAMCM_Prefetchall }

    SEMC IP command for SDARM.

```

Driver version

- #define **FSL_SEMC_DRIVER_VERSION** (MAKE_VERSION(2, 7, 0))
SEMC driver version.

SEMC Initialization and De-initialization

- void **SEMC_GetDefaultConfig** (semc_config_t *config)
Gets the SEMC default basic configuration structure.
- void **SEMC_Init** (SEMC_Type *base, semc_config_t *configure)
Initializes SEMC.
- void **SEMC_Deinit** (SEMC_Type *base)
Deinitializes the SEMC module and gates the clock.

SEMC Configuration Operation For Each Memory Type

- status_t **SEMC_ConfigureSDRAM** (SEMC_Type *base, semc_sdram_cs_t cs, semc_sdram_config_t *config, uint32_t clkSrc_Hz)
Configures SDRAM controller in SEMC.
- status_t **SEMC_ConfigureNAND** (SEMC_Type *base, semc_nand_config_t *config, uint32_t clkSrc_Hz)
Configures NAND controller in SEMC.
- status_t **SEMC_ConfigureNOR** (SEMC_Type *base, semc_nor_config_t *config, uint32_t clkSrc_Hz)
Configures NOR controller in SEMC.
- status_t **SEMC_ConfigureSRAMWithChipSelection** (SEMC_Type *base, semc_sram_cs_t cs, semc_sram_config_t *config, uint32_t clkSrc_Hz)
Configures SRAM controller in SEMC.

- **status_t SEMC_ConfigureSRAM** (SEMC_Type *base, semc_sram_config_t *config, uint32_t clkSrc_Hz)

Configures SRAM controller in SEMC.
- **status_t SEMC_ConfigureDBI** (SEMC_Type *base, semc_dbi_config_t *config, uint32_t clkSrc_Hz)

Configures DBI controller in SEMC.

SEMC Interrupt Operation

- static void **SEMC_EnableInterrupts** (SEMC_Type *base, uint32_t mask)

Enables the SEMC interrupt.
- static void **SEMC_DisableInterrupts** (SEMC_Type *base, uint32_t mask)

Disables the SEMC interrupt.
- static bool **SEMC_GetStatusFlag** (SEMC_Type *base)

Gets the SEMC status.
- static void **SEMC_ClearStatusFlags** (SEMC_Type *base, uint32_t mask)

Clears the SEMC status flag state.

SEMC Memory Access Operation

- static bool **SEMC_IsInIdle** (SEMC_Type *base)

Check if SEMC is in idle.
- **status_t SEMC_SendIPCommand** (SEMC_Type *base, semc_mem_type_t memType, uint32_t address, uint32_t command, uint32_t write, uint32_t *read)

SEMC IP command access.
- static uint16_t **SEMC_BuildNandIPCommand** (uint8_t userCommand, semc_ipcmd_nand_addrmode_t addrMode, semc_ipcmd_nand_cmdmode_t cmdMode)

Build SEMC IP command for NAND.
- static bool **SEMC_IsNandReady** (SEMC_Type *base)

Check if the NAND device is ready.
- **status_t SEMC_IPCommandNandWrite** (SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory write through IP command.
- **status_t SEMC_IPCommandNandRead** (SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NAND device memory read through IP command.
- **status_t SEMC_IPCommandNorWrite** (SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory write through IP command.
- **status_t SEMC_IPCommandNorRead** (SEMC_Type *base, uint32_t address, uint8_t *data, uint32_t size_bytes)

SEMC NOR device memory read through IP command.

37.4 Data Structure Documentation

37.4.1 struct semc_sdram_config_t

1. The memory size in the configuration is in the unit of KB. So memsize_kbytes should be set as 2^2 , 2^3 , 2^4 .etc which is base 2KB exponential function. Take refer to BR0~BR3 register in RM for details.

2. The prescalePeriod_N16Cycle is in unit of 16 clock cycle. It is a exception for prescaleTimer_n16cycle = 0, it means the prescaler timer period is $256 * 16$ clock cycles. For prescaleIf prescalerTimer_n16cycle not equal to 0, The prescaler timer period is prescalePeriod_N16Cycle * 16 clock cycles. idleTimeout_NprescalePeriod, refreshUrgThreshold_NprescalePeriod, refreshPeriod_NprescalePeriod are similar to prescalePeriod_N16Cycle.

Data Fields

- **semc_iomux_pin csxPinMux**
CS pin mux.
- **uint32_t address**
The base address.
- **uint32_t memsize_kbytes**
The memory size in unit of kbytes.
- **smec_port_size_t portSize**
Port size.
- **sem_sdram_burst_len_t burstLen**
Burst length.
- **semc_sdram_column_bit_num_t columnAddrBitNum**
Column address bit number.
- **semc_caslatency_t casLatency**
CAS latency.
- **uint8_t tPrecharge2Act_Ns**
Precharge to active wait time in unit of nanosecond.
- **uint8_t tAct2ReadWrite_Ns**
Act to read/write wait time in unit of nanosecond.
- **uint8_t tRefreshRecovery_Ns**
Refresh recovery time in unit of nanosecond.
- **uint8_t tWriteRecovery_Ns**
write recovery time in unit of nanosecond.
- **uint8_t tCkeOff_Ns**
CKE off minimum time in unit of nanosecond.
- **uint8_t tAct2Precharge_Ns**
Active to precharge in unit of nanosecond.
- **uint8_t tSelfRefRecovery_Ns**
Self refresh recovery time in unit of nanosecond.
- **uint8_t tRefresh2Refresh_Ns**
Refresh to refresh wait time in unit of nanosecond.
- **uint8_t tAct2Act_Ns**
Active to active wait time in unit of nanosecond.
- **uint32_t tPrescalePeriod_Ns**
*Prescaler timer period should not be larger than $256 * 16 * \text{clock cycle}$.*
- **uint32_t tIdleTimeout_Ns**
Idle timeout in unit of prescale time period.
- **uint32_t refreshPeriod_nsPerRow**
*Refresh timer period like $64\text{ms} * 1000000 / 8192$.*
- **uint32_t refreshUrgThreshold**
Refresh urgent threshold.
- **uint8_t refreshBurstLen**

- *Refresh burst length.*
• `uint8_t delayChain`
Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.
- `uint8_t autorefreshTimes`
Auto Refresh cycles times.

Field Documentation

(1) `semc_iomux_pin semc_sdram_config_t::csxPinMux`

The kSEMC_MUXA8 is not valid in sdram pin mux setting.

- (2) `uint32_t semc_sdram_config_t::address`
- (3) `uint32_t semc_sdram_config_t::memsize_kbytes`
- (4) `smec_port_size_t semc_sdram_config_t::portSize`
- (5) `sem_sdram_burst_len_t semc_sdram_config_t::burstLen`
- (6) `semc_sdram_column_bit_num_t semc_sdram_config_t::columnAddrBitNum`
- (7) `semc_caslatency_t semc_sdram_config_t::casLatency`
- (8) `uint8_t semc_sdram_config_t::tPrecharge2Act_Ns`
- (9) `uint8_t semc_sdram_config_t::tAct2ReadWrite_Ns`
- (10) `uint8_t semc_sdram_config_t::tRefreshRecovery_Ns`
- (11) `uint8_t semc_sdram_config_t::tWriteRecovery_Ns`
- (12) `uint8_t semc_sdram_config_t::tCkeOff_Ns`
- (13) `uint8_t semc_sdram_config_t::tAct2Prechage_Ns`
- (14) `uint8_t semc_sdram_config_t::tSelfRefRecovery_Ns`
- (15) `uint8_t semc_sdram_config_t::tRefresh2Refresh_Ns`
- (16) `uint8_t semc_sdram_config_t::tAct2Act_Ns`
- (17) `uint32_t semc_sdram_config_t::tPrescalePeriod_Ns`
- (18) `uint32_t semc_sdram_config_t::tIdleTimeout_Ns`
- (19) `uint32_t semc_sdram_config_t::refreshPeriod_nsPerRow`
- (20) `uint32_t semc_sdram_config_t::refreshUrgThreshold`
- (21) `uint8_t semc_sdram_config_t::refreshBurstLen`
- (22) `uint8_t semc_sdram_config_t::delayChain`
- (23) `uint8_t semc_sdram_config_t::autofreshTimes`

37.4.2 struct semc_nand_timing_config_t

Data Fields

- `uint8_t tCeSetup_Ns`
CE setup time: tCS.

- `uint8_t tCeHold_Ns`
CE hold time: tCH.
- `uint8_t tCeInterval_Ns`
CE interval time: tCEITV.
- `uint8_t tWeLow_Ns`
WE low time: tWP.
- `uint8_t tWeHigh_Ns`
WE high time: tWH.
- `uint8_t tReLow_Ns`
RE low time: tRP.
- `uint8_t tReHigh_Ns`
RE high time: tREH.
- `uint8_t tTurnAround_Ns`
Turnaround time for async mode: tTA.
- `uint8_t tWehigh2Below_Ns`
WE# high to RE# wait time: tWHR.
- `uint8_t tRehigh2Below_Ns`
RE# high to WE# low wait time: tRHW.
- `uint8_t tAle2WriteStart_Ns`
ALE to write start wait time: tADL.
- `uint8_t tReady2Below_Ns`
Ready to RE# low min wait time: tRR.
- `uint8_t tWehigh2Busy_Ns`
WE# high to busy wait time: tWB.

Field Documentation

- (1) `uint8_t semc_nand_timing_config_t::tCeSetup_Ns`
- (2) `uint8_t semc_nand_timing_config_t::tCeHold_Ns`
- (3) `uint8_t semc_nand_timing_config_t::tCeInterval_Ns`
- (4) `uint8_t semc_nand_timing_config_t::tWeLow_Ns`
- (5) `uint8_t semc_nand_timing_config_t::tWeHigh_Ns`
- (6) `uint8_t semc_nand_timing_config_t::tReLow_Ns`
- (7) `uint8_t semc_nand_timing_config_t::tReHigh_Ns`
- (8) `uint8_t semc_nand_timing_config_t::tTurnAround_Ns`
- (9) `uint8_t semc_nand_timing_config_t::tWehigh2Below_Ns`
- (10) `uint8_t semc_nand_timing_config_t::tRehigh2Below_Ns`
- (11) `uint8_t semc_nand_timing_config_t::tAle2WriteStart_Ns`
- (12) `uint8_t semc_nand_timing_config_t::tReady2Below_Ns`
- (13) `uint8_t semc_nand_timing_config_t::tWehigh2Busy_Ns`

37.4.3 struct semc_nand_config_t**Data Fields**

- `semc_iomux_pin cePinMux`
The CE pin mux setting.
- `uint32_t axiAddress`
The base address for AXI nand.
- `uint32_t axiMemsize_kbytes`
The memory size in unit of kbytes for AXI nand.
- `uint32_t ipgAddress`
The base address for IPG nand .
- `uint32_t ipgMemsize_kbytes`
The memory size in unit of kbytes for IPG nand.
- `semc_rdy_polarity_t rdyactivePolarity`
Wait ready polarity.
- `bool edoModeEnabled`
EDO mode enabled.
- `semc_nand_column_bit_num_t columnAddrBitNum`
Column address bit number.
- `semc_nand_address_option_t arrayAddrOption`
Address option.

- `sem_nand_burst_len_t burstLen`
Burst length.
- `smec_port_size_t portSize`
Port size.
- `semc_nand_timing_config_t * timingConfig`
SEMC nand timing configuration.

Field Documentation

(1) `semc_iomux_pin semc_nand_config_t::cePinMux`

The kSEMCMUXRDY is not valid for CE pin setting.

(2) `uint32_t semc_nand_config_t::axiAddress`

(3) `uint32_t semc_nand_config_t::axiMemsize_kbytes`

(4) `uint32_t semc_nand_config_t::ipgAddress`

(5) `uint32_t semc_nand_config_t::ipgMemsize_kbytes`

(6) `semc_rdy_polarity_t semc_nand_config_t::rdyactivePolarity`

(7) `bool semc_nand_config_t::edoModeEnabled`

(8) `semc_nand_column_bit_num_t semc_nand_config_t::columnAddrBitNum`

(9) `semc_nand_address_option_t semc_nand_config_t::arrayAddrOption`

(10) `sem_nand_burst_len_t semc_nand_config_t::burstLen`

(11) `smec_port_size_t semc_nand_config_t::portSize`

(12) `semc_nand_timing_config_t* semc_nand_config_t::timingConfig`

37.4.4 struct semc_nor_config_t

Data Fields

- `semc_iomux_pin cePinMux`
The CE# pin mux setting.
- `semc_iomux_nora27_pin addr27`
The Addr bit 27 pin mux setting.
- `uint32_t address`
The base address.
- `uint32_t memsize_kbytes`
The memory size in unit of kbytes.
- `uint8_t addrPortWidth`
The address port width.
- `semc_rdy_polarity_t rdyactivePolarity`

- **semc_adv_polarity_t advActivePolarity**
ADV# polarity.
- **semc_norsram_column_bit_num_t columnAddrBitNum**
Column address bit number.
- **semc_addr_mode_t addrMode**
Address mode.
- **sem_norsram_burst_len_t burstLen**
Burst length.
- **smec_port_size_t portSize**
Port size.
- **uint8_t tCeSetup_Ns**
The CE setup time.
- **uint8_t tCeHold_Ns**
The CE hold time.
- **uint8_t tCeInterval_Ns**
CE interval minimum time.
- **uint8_t tAddrSetup_Ns**
The address setup time.
- **uint8_t tAddrHold_Ns**
The address hold time.
- **uint8_t tWeLow_Ns**
WE low time for async mode.
- **uint8_t tWeHigh_Ns**
WE high time for async mode.
- **uint8_t tReLow_Ns**
RE low time for async mode.
- **uint8_t tReHigh_Ns**
RE high time for async mode.
- **uint8_t tTurnAround_Ns**
Turnaround time for async mode.
- **uint8_t tAddr2WriteHold_Ns**
Address to write data hold time for async mode.
- **uint8_t latencyCount**
Latency count for sync mode.
- **uint8_t readCycle**
Read cycle time for sync mode.
- **uint8_t delayChain**
Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

Field Documentation

- (1) `semc_iomux_pin semc_nor_config_t::cePinMux`
- (2) `semc_iomux_nora27_pin semc_nor_config_t::addr27`
- (3) `uint32_t semc_nor_config_t::address`
- (4) `uint32_t semc_nor_config_t::memsize_kbytes`
- (5) `uint8_t semc_nor_config_t::addrPortWidth`
- (6) `semc_rdy_polarity_t semc_nor_config_t::rdyactivePolarity`
- (7) `semc_adv_polarity_t semc_nor_config_t::advActivePolarity`
- (8) `semc_norsram_column_bit_num_t semc_nor_config_t::columnAddrBitNum`
- (9) `semc_addr_mode_t semc_nor_config_t::addrMode`
- (10) `sem_norsram_burst_len_t semc_nor_config_t::burstLen`
- (11) `smec_port_size_t semc_nor_config_t::portSize`
- (12) `uint8_t semc_nor_config_t::tCeSetup_Ns`
- (13) `uint8_t semc_nor_config_t::tCeHold_Ns`
- (14) `uint8_t semc_nor_config_t::tCeInterval_Ns`
- (15) `uint8_t semc_nor_config_t::tAddrSetup_Ns`
- (16) `uint8_t semc_nor_config_t::tAddrHold_Ns`
- (17) `uint8_t semc_nor_config_t::tWeLow_Ns`
- (18) `uint8_t semc_nor_config_t::tWeHigh_Ns`
- (19) `uint8_t semc_nor_config_t::tReLow_Ns`
- (20) `uint8_t semc_nor_config_t::tReHigh_Ns`
- (21) `uint8_t semc_nor_config_t::tTurnAround_Ns`
- (22) `uint8_t semc_nor_config_t::tAddr2WriteHold_Ns`
- (23) `uint8_t semc_nor_config_t::latencyCount`
- (24) `uint8_t semc_nor_config_t::readCycle`
- (25) `uint8_t semc_nor_config_t::delayChain`

37.4.5 struct semc_sram_config_t

- **`semc_iomux_nora27_pin`** *The CE# pin mux setting.*
- **`semc_iomux_nora27_pin addr27`** *The Addr bit 27 pin mux setting.*
- **`uint32_t address`** *The base address.*
- **`uint32_t memsize_kbytes`** *The memory size in unit of kbytes.*
- **`uint8_t addrPortWidth`** *The address port width.*
- **`semc_adv_polarity_t advActivePolarity`** *ADV# polarity 1: active high, 0: active low.*
- **`semc_addr_mode_t addrMode`** *Address mode.*
- **`sem_norsram_burst_len_t burstLen`** *Burst length.*
- **`smec_port_size_t portSize`** *Port size.*
- **`semc_sync_mode_t syncMode`** *Sync mode.*
- **`bool waitEnable`** *Wait enable.*
- **`uint8_t waitSample`** *Wait sample.*
- **`semc_adv_level_control_t advLevelCtrl`** *ADV# level control during address hold state, 1: low, 0: high.*
- **`uint8_t tCeSetup_Ns`** *The CE setup time.*
- **`uint8_t tCeHold_Ns`** *The CE hold time.*
- **`uint8_t tCeInterval_Ns`** *CE interval minimum time.*
- **`uint8_t readHoldTime_Ns`** *read hold time.*
- **`uint8_t tAddrSetup_Ns`** *The address setup time.*
- **`uint8_t tAddrHold_Ns`** *The address hold time.*
- **`uint8_t tWeLow_Ns`** *WE low time for async mode.*
- **`uint8_t tWeHigh_Ns`** *WE high time for async mode.*
- **`uint8_t tReLow_Ns`** *RE low time for async mode.*
- **`uint8_t tReHigh_Ns`** *RE high time for async mode.*
- **`uint8_t tTurnAround_Ns`** *Turnaround time for async mode.*
- **`uint8_t tAddr2WriteHold_Ns`** *Address to write data hold time for async mode.*
- **`uint8_t tWriteSetup_Ns`** *Write data setup time for sync mode.*

- `uint8_t tWriteHold_Ns`
Write hold time for sync mode.
- `uint8_t latencyCount`
Latency count for sync mode.
- `uint8_t readCycle`
Read cycle time for sync mode.
- `uint8_t delayChain`
Delay chain, which adds delays on DQS clock to compensate timings while DQS is faster than read data.

Field Documentation

- (1) `semc_iomux_pin semc_sram_config_t::cePinMux`
- (2) `semc_iomux_nora27_pin semc_sram_config_t::addr27`
- (3) `uint32_t semc_sram_config_t::address`
- (4) `uint32_t semc_sram_config_t::memsize_kbytes`
- (5) `uint8_t semc_sram_config_t::addrPortWidth`
- (6) `semc_adv_polarity_t semc_sram_config_t::advActivePolarity`
- (7) `semc_addr_mode_t semc_sram_config_t::addrMode`
- (8) `sem_norsram_burst_len_t semc_sram_config_t::burstLen`
- (9) `smec_port_size_t semc_sram_config_t::portSize`
- (10) `semc_sync_mode_t semc_sram_config_t::syncMode`
- (11) `bool semc_sram_config_t::waitEnable`
- (12) `uint8_t semc_sram_config_t::waitSample`
- (13) `semc_adv_level_control_t semc_sram_config_t::advLevelCtrl`
- (14) `uint8_t semc_sram_config_t::tCeSetup_Ns`
- (15) `uint8_t semc_sram_config_t::tCeHold_Ns`
- (16) `uint8_t semc_sram_config_t::tCeInterval_Ns`
- (17) `uint8_t semc_sram_config_t::readHoldTime_Ns`
- (18) `uint8_t semc_sram_config_t::tAddrSetup_Ns`
- (19) `uint8_t semc_sram_config_t::tAddrHold_Ns`
- (20) `uint8_t semc_sram_config_t::tWeLow_Ns`
- (21) `uint8_t semc_sram_config_t::tWeHigh_Ns`
- (22) `uint8_t semc_sram_config_t::tReLow_Ns`
- (23) `uint8_t semc_sram_config_t::tReHigh_Ns`
- (24) `uint8_t semc_sram_config_t::tTurnAround_Ns`
- (25) `uint8_t semc_sram_config_t::tAddr2WriteHold_Ns`
- (26) `uint8_t semc_sram_config_t::tWriteSetup_Ns`
- (27) `uint8_t semc_sram_config_t::tWriteHold_Ns`
- (28) `uint8_t semc_sram_config_t::latencyCount`

The CE# pin mux.

- `uint32_t address`

The base address.

- `uint32_t memsize_kbytes`

The memory size in unit of 4kbytes.

- `semc_db1_column_bit_num_t columnAddrBitNum`

Column address bit number.

- `sem_dbi_burst_len_t burstLen`

Burst length.

- `smec_port_size_t portSize`

Port size.

- `uint8_t tCsxSetup_Ns`

The CSX setup time.

- `uint8_t tCsxHold_Ns`

The CSX hold time.

- `uint8_t tWexLow_Ns`

WEX low time.

- `uint8_t tWexHigh_Ns`

WEX high time.

- `uint8_t tRdxLow_Ns`

RDX low time.

- `uint8_t tRdxHigh_Ns`

RDX high time.

- `uint8_t tCsxInterval_Ns`

Write data setup time.

Field Documentation

- (1) `semc_iomux_pin semc_dbi_config_t::csxPinMux`
- (2) `uint32_t semc_dbi_config_t::address`
- (3) `uint32_t semc_dbi_config_t::memsize_kbytes`
- (4) `semc_dbi_column_bit_num_t semc_dbi_config_t::columnAddrBitNum`
- (5) `sem_dbi_burst_len_t semc_dbi_config_t::burstLen`
- (6) `smec_port_size_t semc_dbi_config_t::portSize`
- (7) `uint8_t semc_dbi_config_t::tCsxSetup_Ns`
- (8) `uint8_t semc_dbi_config_t::tCsxHold_Ns`
- (9) `uint8_t semc_dbi_config_t::tWexLow_Ns`
- (10) `uint8_t semc_dbi_config_t::tWexHigh_Ns`
- (11) `uint8_t semc_dbi_config_t::tRdxLow_Ns`
- (12) `uint8_t semc_dbi_config_t::tRdxHigh_Ns`
- (13) `uint8_t semc_dbi_config_t::tCsxInterval_Ns`

37.4.7 struct semc_queuea_weight_struct_t**Data Fields**

- `uint32_t qos`: 4
weight of qos for queue 0 .
- `uint32_t aging`: 4
weight of aging for queue 0.
- `uint32_t slaveHitNoswitch`: 8
weight of read/write no switch for queue 0 .
- `uint32_t slaveHitSwitch`: 8
weight of read/write switch for queue 0.

Field Documentation

- (1) `uint32_t semc_queuea_weight_struct_t::qos`
- (2) `uint32_t semc_queuea_weight_struct_t::aging`
- (3) `uint32_t semc_queuea_weight_struct_t::slaveHitNoswitch`
- (4) `uint32_t semc_queuea_weight_struct_t::slaveHitSwitch`

37.4.8 union semc_queuea_weight_t**Data Fields**

- `semc_queuea_weight_struct_t queueaConfig`
Structure configuration for queueA.
- `uint32_t queueaValue`
Configuration value for queueA which could directly write to the reg.

Field Documentation

- (1) `semc_queuea_weight_struct_t semc_queuea_weight_t::queueaConfig`
- (2) `uint32_t semc_queuea_weight_t::queueaValue`

37.4.9 struct semc_queueb_weight_struct_t**Data Fields**

- `uint32_t qos: 4`
weight of qos for queue 1.
- `uint32_t aging: 4`
weight of aging for queue 1.
- `uint32_t weightPagehit: 8`
weight of page hit for queue 1 only .
- `uint32_t slaveHitNoswitch: 8`
weight of read/write no switch for queue 1.
- `uint32_t bankRotation: 8`
weight of bank rotation for queue 1 only .

Field Documentation

- (1) `uint32_t semc_queueb_weight_struct_t::qos`
- (2) `uint32_t semc_queueb_weight_struct_t::aging`
- (3) `uint32_t semc_queueb_weight_struct_t::weightPagehit`
- (4) `uint32_t semc_queueb_weight_struct_t::slaveHitNoswitch`
- (5) `uint32_t semc_queueb_weight_struct_t::bankRotation`

37.4.10 union semc_queueb_weight_t**Data Fields**

- `semc_queueb_weight_struct_t queuebConfig`
Structure configuration for queueB.
- `uint32_t queuebValue`
Configuration value for queueB which could directly write to the reg.

Field Documentation

- (1) `semc_queueb_weight_struct_t semc_queueb_weight_t::queuebConfig`
- (2) `uint32_t semc_queueb_weight_t::queuebValue`

37.4.11 struct semc_axi_queueweight_t**Data Fields**

- `bool queueaEnable`
Enable queue a.
- `semc_queuea_weight_t queueaWeight`
Weight settings for queue a.
- `bool queuebEnable`
Enable queue b.
- `semc_queueb_weight_t queuebWeight`
Weight settings for queue b.

Field Documentation

- (1) **bool semc_axi_queueweight_t::queueaEnable**
- (2) **semc_queuea_weight_t semc_axi_queueweight_t::queueaWeight**
- (3) **bool semc_axi_queueweight_t::queuebEnable**
- (4) **semc_queueb_weight_t semc_axi_queueweight_t::queuebWeight**

37.4.12 struct semc_config_t

busTimeoutCycles: when busTimeoutCycles is zero, the bus timeout cycle is 255*1024. otherwise the bus timeout cycles is busTimeoutCycles*1024. cmdTimeoutCycles: is used for command execution timeout cycles. it's similar to the busTimeoutCycles.

Data Fields

- **semc_dqs_mode_t dqsMode**
Dummy read strobe mode: use enum in "semc_dqs_mode_t".
- **uint8_t cmdTimeoutCycles**
Command execution timeout cycles.
- **uint8_t busTimeoutCycles**
Bus timeout cycles.
- **semc_axi_queueweight_t queueWeight**
AXI queue weight.

Field Documentation

- (1) **semc_dqs_mode_t semc_config_t::dqsMode**
- (2) **uint8_t semc_config_t::cmdTimeoutCycles**
- (3) **uint8_t semc_config_t::busTimeoutCycles**
- (4) **semc_axi_queueweight_t semc_config_t::queueWeight**

37.5 Macro Definition Documentation**37.5.1 #define FSL_SEMC_DRIVER_VERSION (MAKE_VERSION(2, 7, 0))****37.6 Enumeration Type Documentation****37.6.1 anonymous enum**

Enumerator

kStatus_SEMC_InvalidDeviceType Invalid device type.

kStatus_SEMC_IpCommandExecutionError IP command execution error.

kStatus_SEMC_AxiCommandExecutionError AXI command execution error.
kStatus_SEMC_InvalidMemorySize Invalid memory size.
kStatus_SEMC_InvalidIpCmdDataSize Invalid IP command data size.
kStatus_SEMC_InvalidAddressPortWidth Invalid address port width.
kStatus_SEMC_InvalidDataPortWidth Invalid data port width.
kStatus_SEMC_InvalidSwPinmuxSelection Invalid SW pinmux selection.
kStatus_SEMC_InvalidBurstLength Invalid burst length.
kStatus_SEMC_InvalidColumnAddressBitWidth Invalid column address bit width.
kStatus_SEMC_InvalidBaseAddress Invalid base address.
kStatus_SEMC_InvalidTimerSetting Invalid timer setting.

37.6.2 enum semc_mem_type_t

Enumerator

kSEMC_MemType_SDRAM SDRAM.
kSEMC_MemType_SRAM SRAM.
kSEMC_MemType_NOR NOR.
kSEMC_MemType_NAND NAND.
kSEMC_MemType_8080 1.

37.6.3 enum semc_waitready_polarity_t

Enumerator

kSEMC_LowActive Low active.
kSEMC_HighActive High active.

37.6.4 enum semc_sdram_cs_t

Enumerator

kSEMC_SDRAM_CS0 SEMC SDRAM CS0.
kSEMC_SDRAM_CS1 SEMC SDRAM CS1.
kSEMC_SDRAM_CS2 SEMC SDRAM CS2.
kSEMC_SDRAM_CS3 SEMC SDRAM CS3.

37.6.5 enum semc_sram_cs_t

Enumerator

kSEMC_SRAM_CS0 SEMC SRAM CS0.

kSEMC_SRAM_CS1 SEMC SRAM CS1.
kSEMC_SRAM_CS2 SEMC SRAM CS2.
kSEMC_SRAM_CS3 SEMC SRAM CS3.

37.6.6 enum semc_nand_access_type_t

Enumerator

kSEMC_NAND_ACCESS_BY_AXI Access to NAND flash by AXI bus.
kSEMC_NAND_ACCESS_BY_IPCMD Access to NAND flash by IP bus.

37.6.7 enum semc_interrupt_enable_t

Enumerator

kSEMC_IPCmdDoneInterrupt Ip command done interrupt.
kSEMC_IPCmdErrInterrupt Ip command error interrupt.
kSEMC_AXICmdErrInterrupt AXI command error interrupt.
kSEMC_AXIBusErrInterrupt AXI bus error interrupt.

37.6.8 enum semc_ipcmd_datasize_t

Enumerator

kSEMC_IPcmdDataSize_1bytes The IP command data size 1 byte.
kSEMC_IPcmdDataSize_2bytes The IP command data size 2 byte.
kSEMC_IPcmdDataSize_3bytes The IP command data size 3 byte.
kSEMC_IPcmdDataSize_4bytes The IP command data size 4 byte.

37.6.9 enum semc_refresh_time_t

Enumerator

kSEMC_RefreshThreeClocks The refresh timing with three bus clocks.
kSEMC_RefreshSixClocks The refresh timing with six bus clocks.
kSEMC_RefreshNineClocks The refresh timing with nine bus clocks.

37.6.10 enum semc_caslatency_t

Enumerator

kSEMC_LatencyOne Latency 1.
kSEMC_LatencyTwo Latency 2.
kSEMC_LatencyThree Latency 3.

37.6.11 enum semc_sdram_column_bit_num_t

Enumerator

kSEMC_SdramColumn_12bit 12 bit.
kSEMC_SdramColumn_11bit 11 bit.
kSEMC_SdramColumn_10bit 10 bit.
kSEMC_SdramColumn_9bit 9 bit.
kSEMC_SdramColumn_8bit 8 bit.

37.6.12 enum sem_sdram_burst_len_t

Enumerator

kSEMC_Sdram_BurstLen1 According to ERR050577, Auto-refresh command may possibly fail to be triggered during long time back-to-back write (or read) when SDRAM controller's burst length is greater than 1. Burst length 1
kSEMC_Sdram_BurstLen2 Burst length 2.
kSEMC_Sdram_BurstLen4 Burst length 4.
kSEMC_Sdram_BurstLen8 Burst length 8.

37.6.13 enum semc_nand_column_bit_num_t

Enumerator

kSEMC_NandColum_16bit 16 bit.
kSEMC_NandColum_15bit 15 bit.
kSEMC_NandColum_14bit 14 bit.
kSEMC_NandColum_13bit 13 bit.
kSEMC_NandColum_12bit 12 bit.
kSEMC_NandColum_11bit 11 bit.
kSEMC_NandColum_10bit 10 bit.
kSEMC_NandColum_9bit 9 bit.

37.6.14 enum sem_nand_burst_len_t

Enumerator

kSEMC_Nand_BurstLen1 Burst length 1.
kSEMC_Nand_BurstLen2 Burst length 2.
kSEMC_Nand_BurstLen4 Burst length 4.
kSEMC_Nand_BurstLen8 Burst length 8.
kSEMC_Nand_BurstLen16 Burst length 16.
kSEMC_Nand_BurstLen32 Burst length 32.
kSEMC_Nand_BurstLen64 Burst length 64.

37.6.15 enum semc_norsram_column_bit_num_t

Enumerator

kSEMC_NorColum_12bit 12 bit.
kSEMC_NorColum_11bit 11 bit.
kSEMC_NorColum_10bit 10 bit.
kSEMC_NorColum_9bit 9 bit.
kSEMC_NorColum_8bit 8 bit.
kSEMC_NorColum_7bit 7 bit.
kSEMC_NorColum_6bit 6 bit.
kSEMC_NorColum_5bit 5 bit.
kSEMC_NorColum_4bit 4 bit.
kSEMC_NorColum_3bit 3 bit.
kSEMC_NorColum_2bit 2 bit.

37.6.16 enum sem_norsram_burst_len_t

Enumerator

kSEMC_Nor_BurstLen1 Burst length 1.
kSEMC_Nor_BurstLen2 Burst length 2.
kSEMC_Nor_BurstLen4 Burst length 4.
kSEMC_Nor_BurstLen8 Burst length 8.
kSEMC_Nor_BurstLen16 Burst length 16.
kSEMC_Nor_BurstLen32 Burst length 32.
kSEMC_Nor_BurstLen64 Burst length 64.

37.6.17 enum semc_dbi_column_bit_num_t

Enumerator

kSEMC_Dbi_Colum_12bit 12 bit.
kSEMC_Dbi_Colum_11bit 11 bit.
kSEMC_Dbi_Colum_10bit 10 bit.
kSEMC_Dbi_Colum_9bit 9 bit.
kSEMC_Dbi_Colum_8bit 8 bit.
kSEMC_Dbi_Colum_7bit 7 bit.
kSEMC_Dbi_Colum_6bit 6 bit.
kSEMC_Dbi_Colum_5bit 5 bit.
kSEMC_Dbi_Colum_4bit 4 bit.
kSEMC_Dbi_Colum_3bit 3 bit.
kSEMC_Dbi_Colum_2bit 2 bit.

37.6.18 enum sem_dbi_burst_len_t

Enumerator

kSEMC_Dbi_BurstLen1 Burst length 1.
kSEMC_Dbi_BurstLen2 Burst length 2.
kSEMC_Dbi_Dbi_BurstLen4 Burst length 4.
kSEMC_Dbi_BurstLen8 Burst length 8.
kSEMC_Dbi_BurstLen16 Burst length 16.
kSEMC_Dbi_BurstLen32 Burst length 32.
kSEMC_Dbi_BurstLen64 Burst length 64.

37.6.19 enum semc_iomux_pin

Enumerator

kSEMC_MUXA8 MUX A8 pin.
kSEMC_MUXCSX0 MUX CSX0 pin.
kSEMC_MUXCSX1 MUX CSX1 Pin.
kSEMC_MUXCSX2 MUX CSX2 Pin.
kSEMC_MUXCSX3 MUX CSX3 Pin.
kSEMC_MUXRDY MUX RDY pin.

37.6.20 enum semc_iomux_nora27_pin

Enumerator

kSEMC_MORA27_NONE No NOR/SRAM A27 pin.
kSEMC_NORA27_MUXCSX3 MUX CSX3 Pin.
kSEMC_NORA27_MUXRDY MUX RDY pin.

37.6.21 enum smec_port_size_t

Enumerator

kSEMC_PortSize8Bit 8-Bit port size.
kSEMC_PortSize16Bit 16-Bit port size.
kSEMC_PortSize32Bit 32-Bit port size.

37.6.22 enum semc_addr_mode_t

Enumerator

kSEMC_AddrDataMux SEMC address/data mux mode.
kSEMC_AdvAddrdataMux Advanced address/data mux mode.
kSEMC_AddrDataNonMux Address/data non-mux mode.

37.6.23 enum semc_dqs_mode_t

Enumerator

kSEMC_Loopbackinternal Dummy read strobe loopbacked internally.
kSEMC_Loopbackdqspad Dummy read strobe loopbacked from DQS pad.

37.6.24 enum semc_adv_polarity_t

Enumerator

kSEMC_AdvActiveLow Adv active low.
kSEMC_AdvActiveHigh Adv active high.

37.6.25 enum semc_sync_mode_t

Enumerator

kSEMC_AsyncMode Async mode.*kSEMC_SyncMode* Sync mode.**37.6.26 enum semc_adv_level_control_t**

Enumerator

kSEMC_AdvHigh Adv is high during address hold state.*kSEMC_AdvLow* Adv is low during address hold state.**37.6.27 enum semc_rdy_polarity_t**

Enumerator

kSEMC_RdyActiveLow Adv active low.*kSEMC_RdyActivehigh* Adv active low.**37.6.28 enum semc_ipcmd_nand_addrmode_t**

Enumerator

kSEMC_NANDAM_ColumnRow Address mode: column and row address(5Byte-CA0/CA1/RA0/-RA1/RA2).*kSEMC_NANDAM_ColumnCA0* Address mode: column address only(1 Byte-CA0).*kSEMC_NANDAM_ColumnCA0CA1* Address mode: column address only(2 Byte-CA0/CA1).*kSEMC_NANDAM_RawRA0* Address mode: row address only(1 Byte-RA0).*kSEMC_NANDAM_RawRA0RA1* Address mode: row address only(2 Byte-RA0/RA1).*kSEMC_NANDAM_RawRA0RA1RA2* Address mode: row address only(3 Byte-RA0).**37.6.29 enum semc_ipcmd_nand_cmdmode_t**

Enumerator

kSEMC_NANDCM_Command command.*kSEMC_NANDCM_CommandHold* Command hold.*kSEMC_NANDCM_CommandAddress* Command address.

kSEMC_NANDCM_CommandAddressHold Command address hold.
kSEMC_NANDCM_CommandAddressRead Command address read.
kSEMC_NANDCM_CommandAddressWrite Command address write.
kSEMC_NANDCM_CommandRead Command read.
kSEMC_NANDCM_CommandWrite Command write.
kSEMC_NANDCM_Read Read.
kSEMC_NANDCM_Write Write.

37.6.30 enum semc_nand_address_option_t

Enumerator

kSEMC_NandAddrOption_5byte_CA2RA3 CA0+CA1+RA0+RA1+RA2.
kSEMC_NandAddrOption_4byte_CA2RA2 CA0+CA1+RA0+RA1.
kSEMC_NandAddrOption_3byte_CA2RA1 CA0+CA1+RA0.
kSEMC_NandAddrOption_4byte_CA1RA3 CA0+RA0+RA1+RA2.
kSEMC_NandAddrOption_3byte_CA1RA2 CA0+RA0+RA1.
kSEMC_NandAddrOption_2byte_CA1RA1 CA0+RA0.

37.6.31 enum semc_ipcmd_nor_dbi_t

Enumerator

kSEMC_NORDBICM_Read NOR read.
kSEMC_NORDBICM_Write NOR write.

37.6.32 enum semc_ipcmd_sram_t

Enumerator

kSEMC_SRAMCM_ArrayRead SRAM memory array read.
kSEMC_SRAMCM_ArrayWrite SRAM memory array write.
kSEMC_SRAMCM_RegRead SRAM memory register read.
kSEMC_SRAMCM_RegWrite SRAM memory register write.

37.6.33 enum semc_ipcmd_sdram_t

Enumerator

kSEMC_SDRAMCM_Read SDRAM memory read.

kSEMC_SDRAMCM_Write SDRAM memory write.
kSEMC_SDRAMCM_Modeset SDRAM MODE SET.
kSEMC_SDRAMCM_Active SDRAM active.
kSEMC_SDRAMCM_AutoRefresh SDRAM auto-refresh.
kSEMC_SDRAMCM_SelfRefresh SDRAM self-refresh.
kSEMC_SDRAMCM_Preload SDRAM precharge.
kSEMC_SDRAMCM_Preloadall SDRAM precharge all.

37.7 Function Documentation

37.7.1 void SEMC_GetDefaultConfig (semc_config_t * config)

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

```
semc_config_t config;
SEMC_GetDefaultConfig(&config);
```

Parameters

<i>config</i>	The SEMC configuration structure pointer.
---------------	---

37.7.2 void SEMC_Init (SEMC_Type * base, semc_config_t * configure)

This function ungates the SEMC clock and initializes SEMC. This function must be called before calling any other SEMC driver functions.

Parameters

<i>base</i>	SEMC peripheral base address.
<i>configure</i>	The SEMC configuration structure pointer.

37.7.3 void SEMC_Deinit (SEMC_Type * base)

This function gates the SEMC clock. As a result, the SEMC module doesn't work after calling this function, for some IDE, calling this API may cause the next downloading operation failed. so, please call this API cautiously. Additional, users can using "#define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (1)" to disable the clock control operation in drivers.

Parameters

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

37.7.4 status_t SEMC_ConfigureSDRAM (**SEMC_Type** * *base*, **semc_sram_cs_t** * *cs*, **semc_sram_config_t** * *config*, **uint32_t** *clkSrc_Hz*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>cs</i>	The chip selection.
<i>config</i>	The sdram configuration.
<i>clkSrc_Hz</i>	The SEMC clock frequency.

37.7.5 status_t SEMC_ConfigureNAND (**SEMC_Type** * *base*, **semc_nand_config_t** * *config*, **uint32_t** *clkSrc_Hz*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>config</i>	The nand configuration.
<i>clkSrc_Hz</i>	The SEMC clock frequency.

37.7.6 status_t SEMC_ConfigureNOR (**SEMC_Type** * *base*, **semc_nor_config_t** * *config*, **uint32_t** *clkSrc_Hz*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>config</i>	The nor configuration.
<i>clkSrc_Hz</i>	The SEMC clock frequency.

37.7.7 status_t SEMC_ConfigureSRAMWithChipSelection (**SEMC_Type** * *base*, **semc_sram_cs_t** *cs*, **semc_sram_config_t** * *config*, **uint32_t** *clkSrc_Hz*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>cs</i>	The chip selection.
<i>config</i>	The sram configuration.
<i>clkSrc_Hz</i>	The SEMC clock frequency.

37.7.8 status_t SEMC_ConfigureSRAM (SEMC_Type * *base*, semc_sram_config_t * *config*, uint32_t *clkSrc_Hz*)

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

Parameters

<i>base</i>	SEMC peripheral base address.
<i>config</i>	The sram configuration.
<i>clkSrc_Hz</i>	The SEMC clock frequency.

37.7.9 status_t SEMC_ConfigureDBI (SEMC_Type * *base*, semc_dbi_config_t * *config*, uint32_t *clkSrc_Hz*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>config</i>	The dbi configuration.
<i>clkSrc_Hz</i>	The SEMC clock frequency.

37.7.10 static void SEMC_EnableInterrupts (SEMC_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [semc_interrupt_enable_t](#). For example, to enable the IP command done and error interrupt, do the following.

```
*     SEMC_EnableInterrupts(ENET, kSEMC_IPCmdDoneInterrupt |
    kSEMC_IPCmdErrInterrupt);
```

Parameters

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

37.7.11 static void SEMC_DisableInterrupts (SEMC_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the SEMC interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [semc_interrupt_enable_t](#). For example, to disable the IP command done and error interrupt, do the following.

```
*     SEMC_DisableInterrupts(ENET,
    kSEMC_IPCmdDoneInterrupt | kSEMC_IPCmdErrInterrupt);
*
```

Parameters

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

37.7.12 static bool SEMC_GetStatusFlag (SEMC_Type * *base*) [inline], [static]

This function gets the SEMC interrupts event status. User can use the a logical OR of enumeration member as a mask. See [semc_interrupt_enable_t](#).

Parameters

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

Returns

status flag, use status flag in [semc_interrupt_enable_t](#) to get the related status.

37.7.13 static void SEMC_ClearStatusFlags (SEMC_Type * *base*, uint32_t *mask*) [inline], [static]

The following status register flags can be cleared SEMC interrupt status.

Parameters

<i>base</i>	SEMC base pointer
<i>mask</i>	The status flag mask, a logical OR of enumeration member semc_interrupt_enable_t .

37.7.14 static bool SEMC_IsIdle (SEMC_Type * *base*) [inline], [static]

Parameters

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

Returns

True SEMC is in idle, false is not in idle.

37.7.15 status_t SEMC_SendIPCommand (SEMC_Type * *base*, semc_mem_type_t *memType*, uint32_t *address*, uint32_t *command*, uint32_t *write*, uint32_t * *read*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>memType</i>	SEMC memory type. refer to "semc_mem_type_t"
<i>address</i>	SEMC device address.
<i>command</i>	SEMC IP command. For NAND device, we should use the SEMC_BuildNandIPCommand to get the right nand command. For NOR/DBI device, take refer to "semc_ipcmd_nor_db_t". For SRAM device, take refer to "semc_ipcmd_sram_t". For SDRAM device, take refer to "semc_ipcmd_sdram_t".
<i>write</i>	Data for write access.
<i>read</i>	Data pointer for read data out.

37.7.16 static uint16_t SEMC_BuildNandIPCommand (uint8_t *userCommand*, semc_ipcmd_nand_addrmode_t *addrMode*, semc_ipcmd_nand_cmdmode_t *cmdMode*) [inline], [static]

This function build SEMC NAND IP command. The command is build of user command code, SEMC address mode and SEMC command mode.

Parameters

<i>userCommand</i>	NAND device normal command.
<i>addrMode</i>	NAND address mode. Refer to "semc_ipcmd_nand_addrmode_t".
<i>cmdMode</i>	NAND command mode. Refer to "semc_ipcmd_nand_cmdmode_t".

37.7.17 static bool SEMC_IsNandReady (SEMC_Type * *base*) [inline], [static]

Parameters

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

Returns

True NAND is ready, false NAND is not ready.

37.7.18 status_t SEMC_IPCommandNandWrite (SEMC_Type * *base*, uint32_t *address*, uint8_t * *data*, uint32_t *size_bytes*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>address</i>	SEMC NAND device address.
<i>data</i>	Data for write access.
<i>size_bytes</i>	Data length.

37.7.19 status_t SEMC_IPCommandNandRead (SEMC_Type * *base*, uint32_t *address*, uint8_t * *data*, uint32_t *size_bytes*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>address</i>	SEMC NAND device address.
<i>data</i>	Data pointer for data read out.
<i>size_bytes</i>	Data length.

37.7.20 status_t SEMC_IPCommandNorWrite (**SEMC_Type** * *base*, **uint32_t** *address*, **uint8_t** * *data*, **uint32_t** *size_bytes*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>address</i>	SEMC NOR device address.
<i>data</i>	Data for write access.
<i>size_bytes</i>	Data length.

37.7.21 status_t SEMC_IPCommandNorRead (**SEMC_Type** * *base*, **uint32_t** *address*, **uint8_t** * *data*, **uint32_t** *size_bytes*)

Parameters

<i>base</i>	SEMC peripheral base address.
<i>address</i>	SEMC NOR device address.
<i>data</i>	Data pointer for data read out.
<i>size_bytes</i>	Data length.

Chapter 38

SINC: SINC Filter

38.1 Overview

The MCUXpresso SDK provides a peripheral driver for the SINC Filter module of MCUXpresso SDK devices. The SINC module is a digital filter that converts an external ADC sigma-delta modulator bitstream to a data stream. One SINC module contains 4 channel4 with dedicated modulator clocks, bitstreams, and triggers. Based on features of SINC module, this driver are divided into following function groups:

SINC Generic Functional Control APIs

Functions in this group are used to initialize or de-initialize SINC module, users can use SINC module by only invoking [SINC_Init\(\)](#) with correct configuration.

SINC Generic Low Level Control APIs

Functions in this group are used to enable/disable SINC master or reset SINC module.

Clock Control APIs

Functions in this group are used to control SINC internal clocks.

Channel Functional Control APIs

Functions in this group can be used to set specific channel in the high level method. For example, if only want to set all options(input, protection, FIFO and so on) of channel0, invoking [SINC_SetChannelConfig\(\)](#) with the second parameter as [kSINC_Channel0](#).

Channel Result/FIFO Low Level Control APIs

Functions in this group can be used to set specific channel's result related options.

Channel Conversion Low Level Control APIs

Functions in this group can be used to set specific channel's conversion related options.

Channel Protection Low Level Control APIs

Functions in this group can be used to set specific channel's protection related options.

Channel Debug Low Level Control APIs

Functions in this group can be used to set specific channel's debug options.

Interrupt Control APIs

Functions in this group are used to enable/disable SINC module interrupts.

Data Structures

- struct `sinc_channel_input_option_t`
The structure of channel input options, including input bit settings and input clock settings. [More...](#)
- struct `sinc_channel_conv_option_t`
The structure of channel conversion options, including CIC filter settings, HPF settings, shift settings, bias settings and so on. [More...](#)
- struct `sinc_channel_protection_option_t`
The structure of channel protection options, including limit check settings, short-circuit settings, clock-absence settings, and zero-crossing settings. [More...](#)
- struct `sinc_channel_config_t`
The structure of channel configurations, including channel input option, channel conversion options, channel protection options, and so on. [More...](#)
- struct `sinc_config_t`
The structure of sinc configurations, including clock settings anc channels' settings. [More...](#)

Enumerations

- enum `_sinc_interrupt_enable`
The enumeration of SINC module's interrupts.
- enum `_sinc_interrupt_status`
The enumeration of SINC interrupt status flags.
- enum `sinc_channel_id_t` {
 kSINC_Channel0 = 0U,
 kSINC_Channel1,
 kSINC_Channel2,
 kSINC_Channel3 }
The enumeration of channel id, the sinc module contains 4 channels.
- enum `_sinc_modulator_clock` {
 kSINC_ModClk0 = 1UL,
 kSINC_ModClk1 = 2UL,
 kSINC_ModClk2 = 4UL }
The enumeration of modulator clock name.
- enum `sinc_inputClk_source_t` {
 kSINC_InputClk_SourceMclkOut0 = 0U,
 kSINC_InputClk_SourceMclkOut1 = 1U,
 kSINC_InputClk_SourceMclkOut2 = 2U,
 kSINC_InputClk_SourceExternalModulatorClk = 3U,
 kSINC_InputClk_SourceAdjacentChannelClk = 7U }
The enumeration of input clock.
- enum `sinc_inputClk_edge_t` {

```
kSINC_InputClk_EdgePositive = 1U,
kSINC_InputClk_EdgeNegative,
kSINC_InputClk_EdgeBoth,
kSINC_InputClk_EdgeOddPositive,
kSINC_InputClk_EdgeEvenPositive,
kSINC_InputClk_EdgeOddNegative,
kSINC_InputClk_EdgeEvenNegative }
```

The enumeration of clock edge.

- enum `sinc_inputBit_format_t` {

kSINC_InputBit_FormatExternalBitstream = 0U,
 kSINC_InputBit_FormatExternalManchesterCode,
 kSINC_InputBit_FormatInternal16bitParallelData,
 kSINC_InputBit_FormatInternal32bitSerialData }

The enumeration of input bit format.

- enum `sinc_inputBit_source_t` {

kSINC_InputBit_SourceExternalBitstream = 0U,
 kSINC_InputBit_SourceInternalBitstream = 1U,
 kSINC_InputBit_SourceAdjacentChannel = 3U }

The enumeration of input bit source.

- enum `sinc_conv_trigger_source_t` {

kSINC_ConvTrig_SoftPosEdge = 0x0U,
 kSINC_ConvTrig_SoftHighLevel = 0x4U,
 kSINC_ConvTrig_HardPosEdge = 0x1U,
 kSINC_ConvTrig_HardHighLevel = 0x5U,
 kSINC_ConvTrig_AdjacentChannel = 0x3U }

The enumeration of trigger source.

- enum `sinc_conv_mode_t` {

kSINC_ConvMode_Single = 0U,
 kSINC_ConvMode_Continuous,
 kSINC_ConvMode_Always,
 kSINC_ConvMode_FixedNumber }

The enumeration of conversion mode.

- enum `sinc_pulse_trigger_mux_t` {

```

kSINC_PulseTrigger_Disabled = 0U,
kSINC_PulseTrigger_MuxHighLimitLevelSignal,
kSINC_PulseTrigger_MuxLowLimitLevelSignal,
kSINC_PulseTrigger_MuxHighLowLimitLevelSignal,
kSINC_PulseTrigger_MuxWindowLimitLevelSignal,
kSINC_PulseTrigger_MuxZeroCrossRisingLevelSignal,
kSINC_PulseTrigger_MuxZeroCrossFallingLevelSignal,
kSINC_PulseTrigger_MuxRsLimHighLevelSignal,
kSINC_PulseTrigger_MuxRsLimLowLevelSignal,
kSINC_PulseTrigger_MuxChannelRawInputModBitStream,
kSINC_PulseTrigger_MuxChannelRawInputModClock,
kSINC_PulseTrigger_MuxChannelRecoveredModBitStream,
kSINC_PulseTrigger_MuxChannelRecoveredModClock,
kSINC_PulseTrigger_MuxHighLimitPulseSignal,
kSINC_PulseTrigger_MuxLowLimitPulseSignal,
kSINC_PulseTrigger_MuxLimitPulseSignal,
kSINC_PulseTrigger_MuxWindowLimitPulseSignal,
kSINC_PulseTrigger_MuxHighLowLimitPulseSignal,
kSINC_PulseTrigger_MuxZeroCrossRisePulseSignal,
kSINC_PulseTrigger_MuxZeroCrossFallPulseSignal,
kSINC_PulseTrigger_MuxZeroCrossRiseFallPulseSignal,
kSINC_PulseTrigger_MuxFifoWatermarkOkPulseSignal,
kSINC_PulseTrigger_MuxFifoOverflowPulseSignal,
kSINC_PulseTrigger_MuxFifoUnderflowPulseSignal,
kSINC_PulseTrigger_MuxFifoEmptyPulseSignal,
kSINC_PulseTrigger_MuxClockMonitorAssertPulseSignal,
kSINC_PulseTrigger_MuxShortCircuitAssertPulseSignal,
kSINC_PulseTrigger_MuxSaturationPulseSignal,
kSINC_PulseTrigger_MuxConversionCompletePulseSignal }

```

The enumeration of pulse trigger mux.

- enum `sinc_zero_cross_operate_mode_t` {

kSINC_ZCD_BothRiseAndFall = 0U,

kSINC_ZCD_OnlyFall,

kSINC_ZCD_OnlyRise,

kSINC_ZCD_Disabled = 0xFU }

The enumeration of zero cross detector operate mode.

- enum `sinc_primary_filter_order_t` {

kSINC_PF_FastSinc = 0U,

kSINC_PF_FirstOrder,

kSINC_PF_SecondOrder,

kSINC_PF_ThirdOrder }

The enumeration of primary filter order.

- enum `sinc_clock_prescale_t` {

```
kSINC_ClkPrescale1 = 0UL,
kSINC_ClkPrescale2,
kSINC_ClkPrescale4,
kSINC_ClkPrescale8 }
```

The enumeration of clock prescale that specify the clock divider ratio for the modulator clock.

- enum [sinc_primary_filter_shift_direction_t](#) {
 kSINC_PF_ShiftRight = 0U,
 kSINC_PF_ShiftLeft = 1U << 4U }

The enumeration of primary filter shift direction.

- enum [sinc_primary_filter_bias_sign_t](#) {
 kSINC_PF_BiasPositive = 0UL,
 kSINC_PF_BiasNegative }

The enumeration of primary filer bias sign.

- enum [sin_primary_filter_hpf_alpha_coeff_t](#) {
 kSINC_PF_HPFAlphaCoeff0 = 0UL,
 kSINC_PF_HPFAlphaCoeff1,
 kSINC_PF_HPFAlphaCoeff2,
 kSINC_PF_HPFAlphaCoeff3,
 kSINC_PF_HPFAlphaCoeff4,
 kSINC_PF_HPFAlphaCoeff5,
 kSINC_PF_HPFAlphaCoeff6,
 kSINC_PF_HPFAlphaCoeff7,
 kSINC_PF_HPFAlphaCoeff8,
 kSINC_PF_HPFAlphaCoeff9,
 kSINC_PF_HPFAlphaCoeff10,
 kSINC_PF_HPFAlphaCoeff11,
 kSINC_PF_HPFAlphaCoeff12,
 kSINC_PF_HPFAlphaCoeff13,
 kSINC_PF_HPFAlphaCoeff14,
 kSINC_PF_HPFAlphaCoeff15 }

The enumeration of HPF DC remover Alpha coefficient.

- enum [sinc_inputBit_delay_t](#) {

```
kSINC_InputBit_DelayDisabled = 0U,
kSINC_InputBit_Delay1ClkCycle,
kSINC_InputBit_Delay2ClkCycle,
kSINC_InputBit_Delay3ClkCycle,
kSINC_InputBit_Delay4ClkCycle,
kSINC_InputBit_Delay5ClkCycle,
kSINC_InputBit_Delay6ClkCycle,
kSINC_InputBit_Delay7ClkCycle,
kSINC_InputBit_Delay8ClkCycle,
kSINC_InputBit_Delay9ClkCycle,
kSINC_InputBit_Delay10ClkCycle,
kSINC_InputBit_Delay11ClkCycle,
kSINC_InputBit_Delay12ClkCycle,
kSINC_InputBit_Delay13ClkCycle,
kSINC_InputBit_Delay14ClkCycle,
kSINC_InputBit_Delay15ClkCycle }
```

The enumeration of input bit delay.

- enum `sinc_scd_operate_mode_t` {

kSINC_Scd_OperateAtChannelEnabled = 0U,

kSINC_Scd_OperateAtConversion,

kSINC_Scd_OperateDisabled = 0xFU }

The enumeration of short-circuit detector operate mode.

- enum `sinc_scd_option_t` {

kSINC_Scd_DetectRepeating0And1 = 0U,

kSINC_Scd_DetectRepeatingOnly1,

kSINC_Scd_DetectRepeatingOnly0 }

The enumeration of short-circuit detector option.

- enum `sinc_limit_detector_mode_t` {

kSINC_Lmt_BothHighAndLowLimit = 0x94U,

kSINC_Lmt_OnlyHighLimit = 0x91U,

kSINC_Lmt_OnlyLowLimit = 0x86U,

kSINC_Lmt_WindowedValue = 0x8BU,

kSINC_Lmt_Disabled = 0x0U }

The mode of limit detector.

- enum `sinc_cad_threshold_t` {

```
kSINC_Cad_Disabled = 0U,
kSINC_Cad_Count1ClkCycle,
kSINC_Cad_Count2ClkCycle,
kSINC_Cad_Count3ClkCycle,
kSINC_Cad_Count4ClkCycle,
kSINC_Cad_Count5ClkCycle,
kSINC_Cad_Count6ClkCycle,
kSINC_Cad_Count7ClkCycle,
kSINC_Cad_Count8ClkCycle,
kSINC_Cad_Count9ClkCycle,
kSINC_Cad_Count10ClkCycle,
kSINC_Cad_Count11ClkCycle,
kSINC_Cad_Count12ClkCycle,
kSINC_Cad_Count13ClkCycle,
kSINC_Cad_Count14ClkCycle,
kSINC_Cad_Count15ClkCycle }
```

The enumeration of clock-absence threshold.

- enum `sinc_result_data_format_t` {

kSINC_LeftJustifiedSigned = 0U,

kSINC_LeftJustifiedUnsigned }

The enumeration of result data format.

- enum `sinc_debug_output_t` {

kSINC_Debug_PfFinalData = 0U,

kSINC_Debug_OffsetData,

kSINC_Debug_PfShiftedData,

kSINC_Debug_HpfData,

kSINC_Debug_CicRawData,

kSINC_Debug_ScdHistoricalData,

kSINC_Debug_ManchesterDecoderData,

kSINC_Debug_CadData,

kSINC_Debug_FifoEntriesNum,

kSINC_Debug_ParallelSerialConverterStatus }

The enumeration of debug output.

Driver version

- #define `FSL_SINC_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 3)`)

lower_component_name driver version 2.1.3.

SINC Generic Functional Control APIs

- void `SINC_Init` (`SINC_Type *base, const sinc_config_t *config`)

Initialize selected SINC instance, including clock options and channel options.
- void `SINC_Deinit` (`SINC_Type *base`)

De-initialize selected SINC instance.
- void `SINC_GetDefaultConfig` (`sinc_config_t *config`)

Get default configuration.

SINC Generic Low Level Control APIs

- static void **SINC_EnableMaster** (SINC_Type *base, bool enable)
Enable/disable all function blocks enabled in their respective registers.
- static void **SINC_DoSoftwareReset** (SINC_Type *base)
Reset all function blocks(except for the clock blocks), interrupt statuses.
- static void **SINC_DisableDozeMode** (SINC_Type *base, bool disable)
Disable/enable SINC module when the chip enters Doze or Stop mode.

Clock Control APIs

- static bool **SINC_CheckModulatorClockReady** (SINC_Type *base, uint32_t modClkMasks)
Check whether selected modulator clocks are ready.
- static void **SINC_DisableModulatorClockOutput** (SINC_Type *base, uint32_t modClkMasks, bool disable)
Disable/enable modulator clocks' output.
- static void **SINC_SetClkPrescale** (SINC_Type *base, sinc_clock_prescale_t clkPrescale)
Set the clock divider ratio for the modulator clock.
- static void **SINC_SetModulatorClockDivider** (SINC_Type *base, uint32_t modClkDivider)
Set modulator clock divider value.

Channel Functional Control APIs

- void **SINC_SetChannelConfig** (SINC_Type *base, sinc_channel_id_t chId, sinc_channel_config_t *chConfig)
Set channel configurations, including input options, conversion options and protection options.
- void **SINC_SetChannelInputOption** (SINC_Type *base, sinc_channel_id_t chId, sinc_channel_input_option_t *chInputOption)
Set channel input options, including input bit format, input bit source, input bit delay, input clock source, input clock edge.
- void **SINC_SetChannelConversionOption** (SINC_Type *base, sinc_channel_id_t chId, sinc_channel_conv_option_t *chConvOption)
Set channel conversion options, including conversion mode, trigger source, and primary filter settings.
- void **SINC_SetChannelProtectionOption** (SINC_Type *base, sinc_channel_id_t chId, sinc_channel_protection_option_t *chProtection)
Set channel protection options, including limit check, short-circuit detector, clock-absence detector, and zero-crossing detector.

Channel Result/FIFO Low Level Control APIs

- static uint32_t **SINC_ReadChannelResultData** (SINC_Type *base, sinc_channel_id_t chId)
Read selected channel's result data.
- static void **SINC_EnableChannelFIFO** (SINC_Type *base, sinc_channel_id_t chId, bool enable)
Enable/disable FIFO transfers for the primary filter.
- static void **SINC_SetChannelFifoWatermark** (SINC_Type *base, sinc_channel_id_t chId, uint8_t fifoWaterMark)
Set the FIFO watermark.
- static void **SINC_EnableChannel** (SINC_Type *base, sinc_channel_id_t chId, bool enable)
Enable/disable selected channel.

- static void **SINC_EnableChannelPrimaryDma** (SINC_Type *base, **sinc_channel_id_t** chId, bool enable)

Enable/disable selected channel's primary DMA transfers when the channel's FIFO exceeds its watermark.
- static void **SINC_SetChannelResultDataFormat** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_result_data_format_t** dataFormat)

Set selected channel's result data format.
- static uint8_t **SINC_GetChannelFifoCount** (SINC_Type *base, **sinc_channel_id_t** chId)

Get the number of remaining data entries in the FIFO.
- static bool **SINC_CheckChannelResultDataReady** (SINC_Type *base, **sinc_channel_id_t** chId)

Check whether the data in selected channel's result data register is stable when FIFO is disabled.
- static bool **SINC_CheckChannelFifoEmpty** (SINC_Type *base, **sinc_channel_id_t** chId)

Check whether selected channel's FIFO is empty.

Channel Conversion Low Level Control APIs

- static void **SINC_AffirmChannelSoftwareTrigger** (SINC_Type *base, uint32_t chMask)

Trigger selected channel's conversion.
- static void **SINC_NegateChannelSoftwareTrigger** (SINC_Type *base, uint32_t chMask)

Negate the trigger of selected channel.
- static void **SINC_SetChannelConversionMode** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_conv_mode_t** mode)

Set selected channel's conversion mode.
- static void **SINC_SetChannelTriggerSource** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_conv_trigger_source_t** triggerSource)

Set selected channel's trigger source.
- static void **SINC_SetChannelMultipurposeData** (SINC_Type *base, **sinc_channel_id_t** chId, uint32_t data)

Set multipurpose data to selected channel.
- static void **SINC_SetChannelPfOrder** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_primary_filter_order_t** pfOrder)

Set selected channel's PF order.
- static void **SINC_SetChannelPfOsr** (SINC_Type *base, **sinc_channel_id_t** chId, uint16_t pfOsr)

Set selected channel's PF over sample rate.
- static void **SINC_SetChannelPfHpfAlphaCoeff** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_primary_filter_hpf_alpha_coeff_t** pfHpfAlphaCoeff)

Set selected channel's HPF DC remover Alpha coefficient.
- static void **SINC_SetChannelPfShiftConfig** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_primary_filter_shift_direction_t** pfShiftDirection, uint8_t pfShiftBitsNum)

Set the value that shifts the PF data for the correct 24-bit precision.
- static void **SINC_SetChannelPfBiasConfig** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_primary_filter_bias_sign_t** pfBiasSign, uint32_t pfBiasValue)

Set the bias offset for the selected channel's PF.
- static void **SINC_EnableChannelPrimaryFilter** (SINC_Type *base, **sinc_channel_id_t** chId, bool enable)

Enable/disable selected channel's primary filter.
- static bool **SINC_CheckChannelParallelSerialDataReady** (SINC_Type *base, **sinc_channel_id_t** chId)

Check whether selected channel's multipurpose data is ready to write parallel or serial data.

- static bool **SINC_CheckChannelPrimaryCICSaturation** (SINC_Type *base, **sinc_channel_id_t** chId)
Check whether primary CIC filter saturation occurred.
- static bool **SINC_CheckChannelHPFSaturation** (SINC_Type *base, **sinc_channel_id_t** chId)
Check whether HPF saturation occurred.
- static bool **SINC_CheckChannelShiftSaturation** (SINC_Type *base, **sinc_channel_id_t** chId)
Check whether Shift saturation occurred.
- static bool **SINC_CheckChannelBiasSaturation** (SINC_Type *base, **sinc_channel_id_t** chId)
Check whether bias saturation occurred.
- uint8_t **SINC_GetChannelConversionCount** (SINC_Type *base, **sinc_channel_id_t** chId)
Get selected channel's number of conversions.
- static bool **SINC_CheckChannelConvProgress** (SINC_Type *base, **sinc_channel_id_t** chId)
Check whether the selected channel is in conversion.
- static bool **SINC_CheckChannelReadyForConv** (SINC_Type *base, **sinc_channel_id_t** chId)
Check whether the selected channel is ready for conversion.

Channel Protection Low Level Control APIs

- static void **SINC_SetChannelLowLimitThreshold** (SINC_Type *base, **sinc_channel_id_t** chId, uint32_t lowLimitThreshold)
Set selected channel's low-limit threshold value.
- static void **SINC_SetChannelHighLimitThreshold** (SINC_Type *base, **sinc_channel_id_t** chId, uint32_t highLimitThreshold)
Set selected channel's high-limit threshold value.
- static void **SINC_SetChannelLimitDetectorMode** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_limit_detector_mode_t** mode)
Set selected channel's limit detector mode.
- static void **SINC_EnableChannelHighLimitBreakSignal** (SINC_Type *base, **sinc_channel_id_t** chId, bool enable)
Enable/disable selected channel's high limit break signal.
- static void **SINC_EnableChannelWindowLimitBreakSignal** (SINC_Type *base, **sinc_channel_id_t** chId, bool enable)
Enable/disable selected channel's window limit break signal.
- static void **SINC_EnableChannelLowLimitBreakSignal** (SINC_Type *base, **sinc_channel_id_t** chId, bool enable)
Enable/disable selected channel's low limit break signal.
- static void **SINC_SetChannelScdOperateMode** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_scd_operate_mode_t** opMode)
Set selected channel's short-circuit detector operate mode.
- static void **SINC_SetChannelScdLimitThreshold** (SINC_Type *base, **sinc_channel_id_t** chId, uint8_t u8ScdLimitThreshold)
Set selected channel's Scd limit threshold.
- static void **SINC_SetChannelScdOption** (SINC_Type *base, **sinc_channel_id_t** chId, **sinc_scd_option_t** option)
Set selected channel's SDC option.
- static void **SINC_EnableChannelScdBreakSignal** (SINC_Type *base, **sinc_channel_id_t** chId, bool enable)
Enable/disable the automatic assertion of the BREAK_SCD signal when SINC detects an SCD event on the selected channel.

- static void [SINC_SetChannelCadLimitThreshold](#) (SINC_Type *base, [sinc_channel_id_t](#) chId, [sinc_cad_threshold_t](#) cadLimitThreshold)
Set the threshold value for the CAD counter.
- static void [SINC_EnableChannelCadBreakSignal](#) (SINC_Type *base, [sinc_channel_id_t](#) chId, bool enable)
Enable/disable the automatic assertion of the BREAK_CAD signal when SINC detects a CAD event on the associated channel.
- static void [SINC_SetChannelZcdOperateMode](#) (SINC_Type *base, [sinc_channel_id_t](#) chId, [sinc_zero_cross_operate_mode_t](#) opMode)
Set selected channel's zero-crossing detector operate mode.

Channel Debug Low Level Control APIs

- static void [SINC_SetChannelPulseTriggerMux](#) (SINC_Type *base, [sinc_channel_id_t](#) chId, [sinc_pulse_trigger_mux_t](#) pulseTrigMux)
Set selected channel's pulse trigger mux.
- static void [SINC_SetChannelDebugOutput](#) (SINC_Type *base, [sinc_channel_id_t](#) chId, [sinc_debug_output_t](#) debugOutput)
Set selected channel's debug output.
- static void [SINC_LatchChannelDebugProcedure](#) (SINC_Type *base, [sinc_channel_id_t](#) chId)
Start selected channel's debug data latch procedure.
- static bool [SINC_CheckChannelDebugDataValid](#) (SINC_Type *base, [sinc_channel_id_t](#) chId)
Check if the selected channel's debug data is valid.
- static uint32_t [SINC_GetChannelDebugData](#) (SINC_Type *base, [sinc_channel_id_t](#) chId)
Return selected channel's the debug data that requested by [SINC_SetChannelDebugOutput\(\)](#).

Interrupt Control APIs

- static void [SINC_EnableInterrupts](#) (SINC_Type *base, uint64_t interruptMasks)
Enable the mask of interrupts, such as channel data ready interrupt, channel limit detect interrupt and so on.
- static void [SINC_DisableInterrupts](#) (SINC_Type *base, uint64_t interruptMasks)
Disable the mask of interrupts, such as channel data ready interrupt, channel limit detect interrupt and so on.
- static uint64_t [SINC_GetInterruptStatus](#) (SINC_Type *base)
Get interrupt status flags.
- static void [SINC_ClearInterruptStatus](#) (SINC_Type *base, uint64_t statusMasks)
Clear selected mask of interrupt status flags.

38.2 Data Structure Documentation

38.2.1 struct [sinc_channel_input_option_t](#)

Data Fields

- [sinc_inputBit_format_t](#) inputBitFormat
Specify input bit format, please refer to [sinc_inputBit_format_t](#).
- [sinc_inputBit_source_t](#) inputBitSource
Specify input bit source, please refer to [sinc_inputBit_source_t](#).

- **sinc_inputBit_delay_t** `inputBitDelay`
Specify input bit delay, please refer to [sinc_inputBit_delay_t](#).
- **sinc_inputClk_source_t** `inputClkSource`
Specify input clock source, please refer to [sinc_inputClk_source_t](#).
- **sinc_inputClk_edge_t** `inputClkEdge`
Specify input clock edge, please refer to [sinc_inputClk_edge_t](#).

Field Documentation

- (1) **sinc_inputBit_format_t** `sinc_channel_input_option_t::inputBitFormat`
- (2) **sinc_inputBit_source_t** `sinc_channel_input_option_t::inputBitSource`
- (3) **sinc_inputBit_delay_t** `sinc_channel_input_option_t::inputBitDelay`
- (4) **sinc_inputClk_source_t** `sinc_channel_input_option_t::inputClkSource`
- (5) **sinc_inputClk_edge_t** `sinc_channel_input_option_t::inputClkEdge`

38.2.2 struct sinc_channel_conv_option_t**Data Fields**

- **sinc_conv_mode_t** `convMode`
Specify conversion mode, please refer to [sinc_conv_mode_t](#).
- **sinc_conv_trigger_source_t** `convTriggerSource`
Specify conversion trigger source, please refer to [sinc_conv_trigger_source_t](#).
- **bool** `enableChPrimaryFilter`
Enable/disable channel's primary filter.
- **sinc_primary_filter_order_t** `pfOrder`
Specify the order of primary filter, please refer to [sinc_primary_filter_order_t](#).
- **uint16_t** `u16pfOverSampleRatio`
Control primary filter's OSR, the minimum permissible value is 3, low value produce unpredictable result, the maximum permissible value depend on PF order and the desired data format, if PF order is third order and data format is signed, the maximum OSR value is 1289, if PF order is third order and data format is unsigned, the maximum OSR value is 1624, otherwise the maximum OSR value is 2047.
- **sin_primary_filter_hpfilter_alpha_coeff_t** `pfHpfAlphaCoeff`
Specify HPF's alpha coeff, please refer to [sin_primary_filter_hpfilter_alpha_coeff_t](#).
- **sinc_primary_filter_shift_direction_t** `pfShiftDirection`
Select shift direction, right or left.
- **uint8_t** `u8pfShiftBitsNum`
Specify the number of bits to shift the data, ranges from 0 to 15.
- **sinc_primary_filter_bias_sign_t** `pfBiasSign`
Select bias sign, please refer to [sinc_primary_filter_bias_sign_t](#).
- **uint32_t** `u32pfBiasValue`
Range from 0 to 0x7FFFFFFL.

Field Documentation

- (1) `sinc_conv_mode_t sinc_channel_conv_option_t::convMode`
- (2) `sinc_conv_trigger_source_t sinc_channel_conv_option_t::convTriggerSource`
- (3) `bool sinc_channel_conv_option_t::enableChPrimaryFilter`
- (4) `sinc_primary_filter_order_t sinc_channel_conv_option_t::pfOrder`
- (5) `uint16_t sinc_channel_conv_option_t::u16pfOverSampleRatio`

Please note that the OSR for equation is u16pfOverSampleRatio + 1

- (6) `sin_primary_filter_hpf_alpha_coeff_t sinc_channel_conv_option_t::pfHpfAlphaCoeff`
- (7) `sinc_primary_filter_shift_direction_t sinc_channel_conv_option_t::pfShiftDirection`
- (8) `uint8_t sinc_channel_conv_option_t::u8pfShiftBitsNum`
- (9) `sinc_primary_filter_bias_sign_t sinc_channel_conv_option_t::pfBiasSign`
- (10) `uint32_t sinc_channel_conv_option_t::u32pfBiasValue`

38.2.3 struct sinc_channel_protection_option_t

Data Fields

- `sinc_limit_detector_mode_t limitDetectorMode`
Specify limit detector mode, please refer to [sinc_limit_detector_mode_t](#).
- `bool bEnableLmtBreakSignal`
Enable/disable limit break signal, the details of break signal is depended on detector mode.
- `uint32_t u32LowLimitThreshold`
Specify the low-limit threshold value, range from 0 to 0xFFFFFFFFUL.
- `uint32_t u32HighLimitThreshold`
Specify the high-limit threshold value, range from 0 to 0xFFFFFFFFUL.
- `sinc_scd_operate_mode_t scdOperateMode`
Enable/disable SCD, and set SCD operate timing.
- `uint8_t u8ScdLimitThreshold`
Range from 2 to 255, 0 and 1 are prohibited.
- `sinc_scd_option_t scdOption`
Specify SCD options, please refer to [sinc_scd_option_t](#).
- `bool bEnableScdBreakSignal`
Enable/disable SCD break signal.
- `sinc_cad_threshold_t cadLimitThreshold`
Specify the threshold value for the CAD counter.
- `bool bEnableCadBreakSignal`
Enable/disable CAD break signal.
- `sinc_zero_cross_operate_mode_t zcdOperateMode`
Specify zero cross detector operate mode.

Field Documentation

- (1) `sinc_limit_detector_mode_t sinc_channel_protection_option_t::limitDetectorMode`
- (2) `bool sinc_channel_protection_option_t::bEnableLmtBreakSignal`
- (3) `uint32_t sinc_channel_protection_option_t::u32LowLimitThreshold`
- (4) `uint32_t sinc_channel_protection_option_t::u32HighLimitThreshold`
- (5) `sinc_scd_operate_mode_t sinc_channel_protection_option_t::scdOperateMode`
- (6) `uint8_t sinc_channel_protection_option_t::u8ScdLimitThreshold`
- (7) `sinc_scd_option_t sinc_channel_protection_option_t::scdOption`
- (8) `bool sinc_channel_protection_option_t::bEnableScdBreakSignal`
- (9) `sinc_cad_threshold_t sinc_channel_protection_option_t::cadLimitThreshold`
- (10) `bool sinc_channel_protection_option_t::bEnableCadBreakSignal`
- (11) `sinc_zero_cross_operate_mode_t sinc_channel_protection_option_t::zcdOperateMode`

38.2.4 struct sinc_channel_config_t**Data Fields**

- `bool bEnableChannel`
Enable/disable channel.
- `bool bEnableFifo`
Enable/disable channel's FIFO.
- `uint8_t u8FifoWaterMark`
Specify the fifo watermark, range from 0 to 15.
- `bool bEnablePrimaryDma`
Used to enable/disable primary DMA.
- `sinc_result_data_format_t dataFormat`
Set channel's result data format, please refer to [sinc_result_data_format_t](#).
- `sinc_channel_input_option_t * chInputOption`
The pointer to [sinc_channel_input_option_t](#) that contains channel input options.
- `sinc_channel_conv_option_t * chConvOption`
The pointer to [sinc_channel_conv_option_t](#) that contains channel conversion options.
- `sinc_channel_protection_option_t * chProtectionOption`
The pointer to [sinc_channel_protection_option_t](#) that contains channel protection options.

Field Documentation

- (1) **bool sinc_channel_config_t::bEnableChannel**
- (2) **bool sinc_channel_config_t::bEnableFifo**
- (3) **uint8_t sinc_channel_config_t::u8FifoWaterMark**
- (4) **bool sinc_channel_config_t::bEnablePrimaryDma**
- (5) **sinc_result_data_format_t sinc_channel_config_t::dataFormat**
- (6) **sinc_channel_input_option_t* sinc_channel_config_t::chInputOption**
- (7) **sinc_channel_conv_option_t* sinc_channel_config_t::chConvOption**
- (8) **sinc_channel_protection_option_t* sinc_channel_config_t::chProtectionOption**

38.2.5 struct sinc_config_t

Data Fields

- **sinc_clock_prescale_t clockPreDivider**
Specify modulator clock pre divider, please refer to [sinc_clock_prescale_t](#).
- **uint32_t modClkDivider**
Range from 2 to 256, 0 and 1 are prohibited, to obtain a 50% duty cycle in the MCLK output, write an even value to.
- **bool disableModClk0Output**
Disable/enable modulator clock0 output.
- **bool disableModClk1Output**
Disable/enable modulator clock1 output.
- **bool disableModClk2Output**
Disable/enable modulator clock2 output.
- **sinc_channel_config_t * channelsConfigArray [SINC_CHANNEL_COUNT]**
*The array that contains 4 elements, and the type of each element is [sinc_channel_config_t](#) *, channelsConfigArray[0] corresponding to channel0, channelsConfigArray[1] corresponding to channel1, channelsConfigArray[2] corresponding to channel2, channelsConfigArray[3] corresponding to channel3, if some channels are not used, the corresponding elements should be set as NULL.*
- **bool disableDozeMode**
Disable/enable SINC module when the chip enters Doze or stop mode.
- **bool enableMaster**
Enable/disable all function blocks of SINC module.

Field Documentation

- (1) `sinc_clock_prescale_t sinc_config_t::clockPreDivider`
- (2) `bool sinc_config_t::disableModClk0Output`
- (3) `bool sinc_config_t::disableModClk1Output`
- (4) `bool sinc_config_t::disableModClk2Output`
- (5) `sinc_channel_config_t* sinc_config_t::channelsConfigArray[SINC_CHANNEL_COUNT]`
- (6) `bool sinc_config_t::disableDozeMode`
- (7) `bool sinc_config_t::enableMaster`

38.3 Macro Definition Documentation

38.3.1 #define FSL_SINC_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

38.4 Enumeration Type Documentation**38.4.1 enum _sinc_interrupt_enable****38.4.2 enum _sinc_interrupt_status****38.4.3 enum sinc_channel_id_t**

Enumerator

- `kSINC_Channel0` Channel 0.
- `kSINC_Channel1` Channel 1.
- `kSINC_Channel2` Channel 2.
- `kSINC_Channel3` Channel 3.

38.4.4 enum _sinc_modulator_clock

Enumerator

- `kSINC_ModClk0` Modulator Clock 0 output.
- `kSINC_ModClk1` Modulator Clock 1 output.
- `kSINC_ModClk2` Modulator Clock 2 output.

38.4.5 enum sinc_inputClk_source_t

Enumerator

- kSINC_InputClk_SourceMclkOut0* MCLK_OUT0 with internal routeback.
- kSINC_InputClk_SourceMclkOut1* MCLK_OUT1 with internal routeback.
- kSINC_InputClk_SourceMclkOut2* MCLK_OUT2 with internal routeback.
- kSINC_InputClk_SourceExternalModulatorClk* External modulator clock dedicated to the selected channel.
- kSINC_InputClk_SourceAdjacentChannelClk* Grouped clock shared with an adjacent channel.

38.4.6 enum sinc_inputClk_edge_t

Enumerator

- kSINC_InputClk_EdgePositive* Positive edge.
- kSINC_InputClk_EdgeNegative* Negative edge.
- kSINC_InputClk_EdgeBoth* Both edges.
- kSINC_InputClk_EdgeOddPositive* Every other odd positive edge.
- kSINC_InputClk_EdgeEvenPositive* Every other even positive edge.
- kSINC_InputClk_EdgeOddNegative* Every other odd negative edge.
- kSINC_InputClk_EdgeEvenNegative* Every other even negative edge.

38.4.7 enum sinc_inputBit_format_t

Enumerator

- kSINC_InputBit_FormatExternalBitstream* External bitstream from the MBIT[n] signal.
- kSINC_InputBit_FormatExternalManchesterCode* External Manchester code.
- kSINC_InputBit_FormatInternal16bitParallelData* Internal 16-bit parallel data from MPDATA register.
- kSINC_InputBit_FormatInternal32bitSerialData* Internal 32-bit serial data from MPDATA.

38.4.8 enum sinc_inputBit_source_t

Enumerator

- kSINC_InputBit_SourceExternalBitstream* External bitstream from the MBIT[n] signal.
- kSINC_InputBit_SourceInternalBitstream* Alternate internal bitstream from the INP[n] signal.
- kSINC_InputBit_SourceAdjacentChannel* Grouped bitstream shared with an adjacent channel.

38.4.9 enum sinc_conv_trigger_source_t

Enumerator

kSINC_ConvTrig_SoftPosEdge Positive edge software trigger.

kSINC_ConvTrig_SoftHighLevel High level software trigger.

kSINC_ConvTrig_HardPosEdge Positive edge hardware trigger.

kSINC_ConvTrig_HardHighLevel High level hardware trigger.

kSINC_ConvTrig_AdjacentChannel Grouped hardware trigger shared with an adjacent channel.

38.4.10 enum sinc_conv_mode_t

Enumerator

kSINC_ConvMode_Single One conversion that follows an edge or level trigger event.

kSINC_ConvMode_Continuous Multiple conversions that follow a triggering event, a new triggering event cancels and restarts conversion.

kSINC_ConvMode_Always Multiple conversions that follow the first triggering event, SINC ignores the next triggering event.

kSINC_ConvMode_FixedNumber Fixed number conversions that follow the first triggering event, a new triggering event cancels and restarts conversion.

38.4.11 enum sinc_pulse_trigger_mux_t

Enumerator

kSINC_PulseTrigger_Disabled Disable pulse trigger output.

kSINC_PulseTrigger_MuxHighLimitLevelSignal Select high limit level signal for pulse trigger output.

kSINC_PulseTrigger_MuxLowLimitLevelSignal Select low limit level signal for pulse trigger output.

kSINC_PulseTrigger_MuxHighLowLimitLevelSignal Select low or high limit level signal for pulse trigger output.

kSINC_PulseTrigger_MuxWindowLimitLevelSignal Select window limit level signal for pulse trigger output.

kSINC_PulseTrigger_MuxZeroCrossRisingLevelSignal Select zero cross rising level signal for pulse trigger output.

kSINC_PulseTrigger_MuxZeroCrossFallingLevelSignal Select zero cross falling level signal for pulse trigger output.

kSINC_PulseTrigger_MuxRsLimHighLevelSignal Select level signal that indicates a high level from an RS flip-flop or a schmitt trigger for pulse trigger output.

kSINC_PulseTrigger_MuxRsLimLowLevelSignal Select level signal that indicates a low level from an RS flip-flop or a schmitt trigger for pulse trigger output.

- kSINC_PulseTrigger_MuxChannelRawInputModBitStream*** Select channel raw input modulator bitstream for pulse trigger output.
- kSINC_PulseTrigger_MuxChannelRawInputModClock*** Select channel raw input modulator clock for pulse trigger output.
- kSINC_PulseTrigger_MuxChannelRecoveredModBitStream*** Select channel output recovered modulator bitstream for pulse trigger output.
- kSINC_PulseTrigger_MuxChannelRecoveredModClock*** Select channel output recovered modulator clock for pulse trigger output.
- kSINC_PulseTrigger_MuxHighLimitPulseSignal*** Select high limit pulse signal for pulse trigger output.
- kSINC_PulseTrigger_MuxLowLimitPulseSignal*** Select low limit pulse signal for pulse trigger output.
- kSINC_PulseTrigger_MuxLimitPulseSignal*** Select the pulse signal that indicates a high/low/window limit for pulse trigger output.
- kSINC_PulseTrigger_MuxWindowLimitPulseSignal*** Select window limit pulse signal for pulse trigger output.
- kSINC_PulseTrigger_MuxHighLowLimitPulseSignal*** Select the pulse signal that indicates a high or low limit for pulse trigger output.
- kSINC_PulseTrigger_MuxZeroCrossRisePulseSignal*** Select zero cross rise pulse signal for trigger output.
- kSINC_PulseTrigger_MuxZeroCrossFallPulseSignal*** Select zero cross fall pulse signal for trigger output.
- kSINC_PulseTrigger_MuxZeroCrossRiseFallPulseSignal*** Select zero cross rise/fall pulse signal for trigger output.
- kSINC_PulseTrigger_MuxFifoWatermarkOkPulseSignal*** Select FIFO watermark OK pulse signal for trigger output.
- kSINC_PulseTrigger_MuxFifoOverflowPulseSignal*** Select FIFO overflow pulse signal.
- kSINC_PulseTrigger_MuxFifoUnderflowPulseSignal*** Select FIFO underflow pulse signal.
- kSINC_PulseTrigger_MuxFifoEmptyPulseSignal*** Select FIFO empty pulse signal.
- kSINC_PulseTrigger_MuxClockMonitorAssertPulseSignal*** Select clock monitor assert pulse signal.
- kSINC_PulseTrigger_MuxShortCircuitAssertPulseSignal*** Select short circuit assert pulse signal.
- kSINC_PulseTrigger_MuxSaturationPulseSignal*** Select saturation pulse signal.
- kSINC_PulseTrigger_MuxConversionCompletePulseSignal*** Select conversion complete pulse signal.

38.4.12 enum sinc_zero_cross_operate_mode_t

Enumerator

- kSINC_ZCD_BothRiseAndFall*** Zero cross detector operate on both rise and fall.
- kSINC_ZCD_OnlyFall*** Zero cross detector operate on fall edge.
- kSINC_ZCD_OnlyRise*** Zero cross detector operate on rise edge.
- kSINC_ZCD_Disabled*** Zero cross detector disabled.

38.4.13 enum sinc_primary_filter_order_t

Enumerator

- kSINC_PF_FastSinc*** Fast sinc filter, ORD is 4.
- kSINC_PF_FirstOrder*** First order filter, ORD is 1.
- kSINC_PF_SecondOrder*** Second order filter, ORD is 2.
- kSINC_PF_ThirdOrder*** Third order filter, ORD is 3.

38.4.14 enum sinc_clock_prescale_t

Enumerator

- kSINC_ClkPrescale1*** No prescale.
- kSINC_ClkPrescale2*** Modulator clock divider ratio is 2.
- kSINC_ClkPrescale4*** Modulator clock divider ratio is 4.
- kSINC_ClkPrescale8*** Modulator clock divider ratio is 8.

38.4.15 enum sinc_primary_filter_shift_direction_t

Enumerator

- kSINC_PF_ShiftRight*** Right shift the raw data.
- kSINC_PF_ShiftLeft*** Left shift the raw data.

38.4.16 enum sinc_primary_filter_bias_sign_t

Enumerator

- kSINC_PF_BiasPositive*** The bias sign is positive.
- kSINC_PF_BiasNegative*** The bias sign is negative.

38.4.17 enum sin_primary_filter_hpf_alpha_coeff_t

Enumerator

- kSINC_PF_HPFAlphaCoeff0*** Disabled HPF.
- kSINC_PF_HPFAlphaCoeff1*** Alpha coefficient = $1 - (2^{-5})$
- kSINC_PF_HPFAlphaCoeff2*** Alpha coefficient = $1 - (2^{-6})$
- kSINC_PF_HPFAlphaCoeff3*** Alpha coefficient = $1 - (2^{-7})$

<i>kSINC_PF_HPFAlphaCoeff4</i>	Alpha coefficient = 1 - (2^-8)
<i>kSINC_PF_HPFAlphaCoeff5</i>	Alpha coefficient = 1 - (2^-9)
<i>kSINC_PF_HPFAlphaCoeff6</i>	Alpha coefficient = 1 - (2^-10)
<i>kSINC_PF_HPFAlphaCoeff7</i>	Alpha coefficient = 1 - (2^-11)
<i>kSINC_PF_HPFAlphaCoeff8</i>	Alpha coefficient = 1 - (2^-12)
<i>kSINC_PF_HPFAlphaCoeff9</i>	Alpha coefficient = 1 - (2^-13)
<i>kSINC_PF_HPFAlphaCoeff10</i>	Alpha coefficient = 1 - (2^-14)
<i>kSINC_PF_HPFAlphaCoeff11</i>	Alpha coefficient = 1 - (2^-15)
<i>kSINC_PF_HPFAlphaCoeff12</i>	Alpha coefficient = 1 - (2^-16)
<i>kSINC_PF_HPFAlphaCoeff13</i>	Alpha coefficient = 1 - (2^-17)
<i>kSINC_PF_HPFAlphaCoeff14</i>	Alpha coefficient = 1 - (2^-18)
<i>kSINC_PF_HPFAlphaCoeff15</i>	Alpha coefficient = 1 - (2^-19)

38.4.18 enum sinc_inputBit_delay_t

Enumerator

<i>kSINC_InputBit_DelayDisabled</i>	Input modulator bitstream delay disabled.
<i>kSINC_InputBit_Delay1ClkCycle</i>	Input modulator bitstream delay 1 PRE_CLK cycle.
<i>kSINC_InputBit_Delay2ClkCycle</i>	Input modulator bitstream delay 2 PRE_CLK cycle.
<i>kSINC_InputBit_Delay3ClkCycle</i>	Input modulator bitstream delay 3 PRE_CLK cycle.
<i>kSINC_InputBit_Delay4ClkCycle</i>	Input modulator bitstream delay 4 PRE_CLK cycle.
<i>kSINC_InputBit_Delay5ClkCycle</i>	Input modulator bitstream delay 5 PRE_CLK cycle.
<i>kSINC_InputBit_Delay6ClkCycle</i>	Input modulator bitstream delay 6 PRE_CLK cycle.
<i>kSINC_InputBit_Delay7ClkCycle</i>	Input modulator bitstream delay 7 PRE_CLK cycle.
<i>kSINC_InputBit_Delay8ClkCycle</i>	Input modulator bitstream delay 8 PRE_CLK cycle.
<i>kSINC_InputBit_Delay9ClkCycle</i>	Input modulator bitstream delay 9 PRE_CLK cycle.
<i>kSINC_InputBit_Delay10ClkCycle</i>	Input modulator bitstream delay 10 PRE_CLK cycle.
<i>kSINC_InputBit_Delay11ClkCycle</i>	Input modulator bitstream delay 11 PRE_CLK cycle.
<i>kSINC_InputBit_Delay12ClkCycle</i>	Input modulator bitstream delay 12 PRE_CLK cycle.
<i>kSINC_InputBit_Delay13ClkCycle</i>	Input modulator bitstream delay 13 PRE_CLK cycle.
<i>kSINC_InputBit_Delay14ClkCycle</i>	Input modulator bitstream delay 14 PRE_CLK cycle.
<i>kSINC_InputBit_Delay15ClkCycle</i>	Input modulator bitstream delay 15 PRE_CLK cycle.

38.4.19 enum scd_operate_mode_t

Enumerator

<i>kSINC_Scd_OperateAtChannelEnabled</i>	SCD operates when the channel is enabled.
<i>kSINC_Scd_OperateAtConversion</i>	SCD operates when the PF is performing a conversion.
<i>kSINC_Scd_OperatedDisabled</i>	Short circuit detect is disabled.

38.4.20 enum sinc_scd_option_t

Enumerator

kSINC_Scd_DetectRepeating0And1 Both repeating 0 and 1 increment the SCD counter.

kSINC_Scd_DetectRepeatingOnly1 Only repeating 1 increment the SCD counter.

kSINC_Scd_DetectRepeatingOnly0 Only repeating 1 increment the SCD counter.

38.4.21 enum sinc_limit_detector_mode_t

Note

The value of each limit detector contains lot of information: bit[1:0]: limit detection options, bit[2]: low limit break signal. bit[3]: window limit break signal. bit[4]: High limit break signal. bit[7]: Enable/disable limit detector.

Enumerator

kSINC_Lmt_BothHighAndLowLimit Limit detector is enabled, and compare the filter sample value to high and low limit, if the value larger than high limit will trigger interrupt or break, and if the value lower than low limit will trigger interrupt or break.

kSINC_Lmt_OnlyHighLimit Limit detector is enabled, and compare the filter sample value to high limit, if the value larger than high limit will trigger interrupt or break.

kSINC_Lmt_OnlyLowLimit Limit detector is enabled, and compare the filter sample value to low limit, if the value lower than low limit will trigger interrupt or break.

kSINC_Lmt_WindowedValue Limit detector is enabled, and compare the filter sample value to high and low limit, if the value higher than low limit and lower than high limit will trigger interrupt or break.

kSINC_Lmt_Disabled Limit detector is disabled.

38.4.22 enum sinc_cad_threshold_t

Enumerator

kSINC_Cad_Disabled Clock absence detector is disabled.

kSINC_Cad_Count1ClkCycle Clock absence detector threshold is 1 clock cycle.

kSINC_Cad_Count2ClkCycle Clock absence detector threshold is 2 clock cycle.

kSINC_Cad_Count3ClkCycle Clock absence detector threshold is 3 clock cycle.

kSINC_Cad_Count4ClkCycle Clock absence detector threshold is 4 clock cycle.

kSINC_Cad_Count5ClkCycle Clock absence detector threshold is 5 clock cycle.

kSINC_Cad_Count6ClkCycle Clock absence detector threshold is 6 clock cycle.

kSINC_Cad_Count7ClkCycle Clock absence detector threshold is 7 clock cycle.

kSINC_Cad_Count8ClkCycle Clock absence detector threshold is 8 clock cycle.

kSINC_Cad_Count9ClkCycle Clock absence detector threshold is 9 clock cycle.

<i>kSINC_Cad_Count10ClkCycle</i>	Clock absence detector threshold is 10 clock cycle.
<i>kSINC_Cad_Count11ClkCycle</i>	Clock absence detector threshold is 11 clock cycle.
<i>kSINC_Cad_Count12ClkCycle</i>	Clock absence detector threshold is 12 clock cycle.
<i>kSINC_Cad_Count13ClkCycle</i>	Clock absence detector threshold is 13 clock cycle.
<i>kSINC_Cad_Count14ClkCycle</i>	Clock absence detector threshold is 14 clock cycle.
<i>kSINC_Cad_Count15ClkCycle</i>	Clock absence detector threshold is 15 clock cycle.

38.4.23 enum sinc_result_data_format_t

Enumerator

<i>kSINC_LeftJustifiedSigned</i>	Left justified, signed.
<i>kSINC_LeftJustifiedUnsigned</i>	Left justified, unsigned.

38.4.24 enum sinc_debug_output_t

Enumerator

<i>kSINC_Debug_PfFinalData</i>	Final data from PF(24 bits).
<i>kSINC_Debug_OffsetData</i>	Offset data(24 bits).
<i>kSINC_Debug_PfShiftedData</i>	Shifted data from the PF(24 bits).
<i>kSINC_Debug_HpfData</i>	DC remover(HPF) data(32 bits).
<i>kSINC_Debug_CicRawData</i>	Raw data from the PF's CIC filter.
<i>kSINC_Debug_ScdHistoricalData</i>	Historical data from SCD.
<i>kSINC_Debug_ManchesterDecoderData</i>	Data from the Manchester decoder.
<i>kSINC_Debug_CadData</i>	Data from CAD.
<i>kSINC_Debug_FifoEntriesNum</i>	Number of available entries in the FIFO.
<i>kSINC_Debug_ParallelSerialConverterStatus</i>	Status of the parallel or serial data converter.

38.5 Function Documentation

38.5.1 void SINC_Init (SINC_Type * *base*, const sinc_config_t * *config*)

Parameters

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

<i>config</i>	The pointer to sinc_config_t structure.
---------------	---

38.5.2 void SINC_Deinit (SINC_Type * *base*)

Parameters

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

38.5.3 void SINC_GetDefaultConfig (sinc_config_t * *config*)

```
* config->clockPreDivider = kSINC_ClkPrescale1;
* config->modClkDivider = 2UL;
* config->disableModClk0Output = false;
* config->disableModClk1Output = false;
* config->disableModClk2Output = false;
*
* config->channelsConfigArray[4] = {NULL, NULL, NULL, NULL};
*
* config->disableDozeMode = false;
* config->enableMaster = false;
*
```

Parameters

<i>config</i>	The pointer to sinc_config_t structure, must not be NULL.
---------------	---

**38.5.4 static void SINC_EnableMaster (SINC_Type * *base*, bool *enable*)
[inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>enable</i>	Used to enable/disable all function blocks: <ul style="list-style-type: none"> • true Enable all function blocks, please note that clock must be configured previously; • false Disable all function blocks.

**38.5.5 static void SINC_DoSoftwareReset (SINC_Type * *base*) [inline],
[static]**

Parameters

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

38.5.6 static void SINC_DisableDozeMode (**SINC_Type** * *base*, **bool** *disable*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>disable</i>	Used to control if module functional when the chip enters Doze and Stop mode: <ul style="list-style-type: none"> • true Disable SINC when the chip enters Doze or Stop mode; • false Enable SINC when the chip enters Doze or stop mode.

38.5.7 static bool SINC_CheckModulatorClockReady (**SINC_Type** * *base*, **uint32_t** *modClkMasks*) [inline], [static]

Note

The result of this APIs means all selected modulator clocks are (not) ready.

Parameters

<i>base</i>	SINC peripheral base address.
<i>modClkMasks</i>	The mask of modulator clocks, please refer to _sinc_modulator_clock .

Return values

<i>true</i>	The input mask of modulator clocks are ready.
<i>false</i>	The input mask of modulator clocks are not ready.

38.5.8 static void SINC_DisableModulatorClockOutput (**SINC_Type** * *base*, **uint32_t** *modClkMasks*, **bool** *disable*) [inline], [static]

Note

By default, modulator clock's output is enabled.

Parameters

<i>base</i>	SINC peripheral base address.
<i>modClkMasks</i>	The mask of modulator clocks, please refer to _sinc_modulator_clock .
<i>disable</i>	Used to enable/disable clock output: <ul style="list-style-type: none">• true Disable modulator clocks' output;• false Enable modulator clocks' output.

38.5.9 static void SINC_SetClkPrescale (**SINC_Type** * *base*, **sinc_clock_prescale_t** *clkPrescale*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>clkPrescale</i>	Clock prescale value, please refer to sinc_clock_prescale_t .

38.5.10 static void SINC_SetModulatorClockDivider (**SINC_Type** * *base*, **uint32_t** *modClkDivider*) [inline], [static]

Note

$\text{IMCLK0} = \text{PRE_CLK} / \text{modClkDivider}$, the minimum clock divider ration is 2.

Parameters

<i>base</i>	SINC peripheral base address.
<i>modClkDivider</i>	Range from 2 to 256, 0 and 1 are prohibited, to obtain a 50% duty cycle in the MCLK output, write an even value to modClkDivider .

38.5.11 void SINC_SetChannelConfig (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_channel_config_t** * *chConfig*)

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>chConfig</i>	Pointer to sinc_channel_config_t structure, must not be NULL.

38.5.12 void SINC_SetChannelInputOption (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_channel_input_option_t** * *chInputOption*)

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>chInputOption</i>	Pointer to sinc_channel_input_option_t structure, must not be NULL.

38.5.13 void SINC_SetChannelConversionOption (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_channel_conv_option_t** * *chConvOption*)

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>chConvOption</i>	Pointer to sinc_channel_conv_option_t structure, must not be NULL.

38.5.14 void SINC_SetChannelProtectionOption (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_channel_protection_option_t** * *chProtection*)

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .

<i>chProtection</i>	Pointer to sinc_channel_protection_option_t , must not be NULL.
---------------------	---

38.5.15 static uint32_t SINC_ReadChannelResultData (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .

Returns

Result data of the selected channel, 24 bits width.

38.5.16 static void SINC_EnableChannelFIFO (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable channl FIFO: <ul style="list-style-type: none">• true Enable channel FIFO.• false Disable channe FIFO.

38.5.17 static void SINC_SetChannelFifoWatermark (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **uint8_t** *fifoWaterMark*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>fifoWaterMark</i>	Specify the fifo watermark, range from 0 to 15.

38.5.18 **static void SINC_EnableChannel(SINC_Type * *base*, sinc_channel_id_t *chId*, bool *enable*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable selected channel: <ul style="list-style-type: none"> • true Enable selected channel; • false Disable selected channel.

**38.5.19 static void SINC_EnableChannelPrimaryDma (SINC_Type * *base*,
sinc_channel_id_t chId, *bool enable*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable primary DMA : <ul style="list-style-type: none"> • true Enable primary DMA; • false Disable primary DMA.

**38.5.20 static void SINC_SetChannelResultDataFormat (SINC_Type * *base*,
sinc_channel_id_t chId, *sinc_result_data_format_t dataFormat*)
[inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>dataFormat</i>	Specify the result data format, please refer to sinc_result_data_format_t .

**38.5.21 static uint8_t SINC_GetChannelFifoCount (SINC_Type * *base*,
sinc_channel_id_t chId) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .

Returns

The number of remaining data entries in the FIFO.

38.5.22 static bool SINC_CheckChannelResultDataReady (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, refer to sinc_channel_id_t for details.

Return values

<i>true</i>	Data in selected channel's result data register is stable.
<i>false</i>	Data in selected channel's result data register is not stable.

38.5.23 static bool SINC_CheckChannelFifoEmpty (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	The id of sinc channel to check.

Return values

<i>true</i>	Selected channel's FIFO is empty.
<i>false</i>	Selected channel's FIFO is not empty.

38.5.24 static void SINC_AffirmChannelSoftwareTrigger (**SINC_Type** * *base*, **uint32_t** *chMask*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chMask</i>	The mask of channels to trigger.

38.5.25 static void SINC_NegateChannelSoftwareTrigger (**SINC_Type** * *base*, **uint32_t** *chMask*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chMask</i>	The mask of channels.

38.5.26 static void SINC_SetChannelConversionMode (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_conv_mode_t** *mode*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>mode</i>	The conversion mode to set, please refer to sinc_conv_mode_t .

38.5.27 static void SINC_SetChannelTriggerSource (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_conv_trigger_source_t** *triggerSource*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .

<i>triggerSource</i>	Trigger source to set, please refer to sinc_conv_trigger_source_t .
----------------------	---

38.5.28 static void SINC_SetChannelMultipurposeData (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **uint32_t** *data*) [inline], [static]

Note

If input bit format is set as ManchesterCode, multipurpose data indicates the Manchester decoder threshold value and is 11 bits width; if input bit format is set as parallel, multipurpose data indicates the parallel 16-bit data and is 16 bits width; if input bit format is set as serial, multipurpose data indicates the serial data and is 32 bits width.

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>data</i>	Multipurpose data to set.

38.5.29 static void SINC_SetChannelPfOrder (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_primary_filter_order_t** *pfOrder*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>pfOrder</i>	Primary filter order to set, please refer to sinc_primary_filter_order_t

38.5.30 static void SINC_SetChannelPfOsr (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **uint16_t** *pfOsr*) [inline], [static]

Note

If PF order is third order and data format is signed, the maximum OSR value is 1289, if PF order is third order and data format is unsigned, the maximum OSR value is 1624, otherwise the maximum OSR value is 2047.

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>pfOsr</i>	Control the channel's PF OSR, the minimum permissible value is 3, low value produce unpredictable result, the maximum permissible value depend on PF order and the desired data format.

**38.5.31 static void SINC_SetChannelPfHpfAlphaCoeff (SINC_Type * *base*,
sinc_channel_id_t chId, *sin_primary_filter_hpf_alpha_coeff_t pfHpfAlphaCoeff*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>pfHpfAlphaCoeff</i>	Specify the HPF alpha coefficient or disable HPF as described in sin_primary_filter_hpf_alpha_coeff_t .

**38.5.32 static void SINC_SetChannelPfShiftConfig (SINC_Type * *base*,
sinc_channel_id_t chId, *sin_primary_filter_shift_direction_t pfShiftDirection*, *uint8_t pfShiftBitsNum*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>pfShiftDirection</i>	Specify the PF shift direction, including right and left.
<i>pfShiftBitsNum</i>	Specify the PF shift value, range from 0 to 15.

**38.5.33 static void SINC_SetChannelPfBiasConfig (SINC_Type * *base*,
sinc_channel_id_t chId, *sin_primary_filter_bias_sign_t pfBiasSign*,
uint32_t pfBiasValue) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>pfBiasSign</i>	Specify the bias sign, please refer to sinc_primary_filter_bias_sign_t for details.
<i>pfBiasValue</i>	The bias value to subtracted from the output of PF shift block, range from 0 to 0x7FFFUL.

38.5.34 static void SINC_EnableChannelPrimaryFilter (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable primary filter: <ul style="list-style-type: none"> • true Enable channel's PF; • false Disable channel's PF.

38.5.35 static bool SINC_CheckChannelParallelSerialDataReady (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, refer to sinc_channel_id_t for details.

Return values

<i>true</i>	Selected channel's multipurpose data is ready to write parallel or serial data.
-------------	---

<i>false</i>	Selected channel's multipurpose data is not ready to write parallel or serial data.
--------------	---

38.5.36 static bool SINC_CheckChannelPrimaryCICSaturation (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, refer to sinc_channel_id_t for details.

Return values

<i>true</i>	Selected channel's primary CIC filter saturation occurred.
<i>false</i>	Selected channel's primary CIC filter saturation did not occurred.

38.5.37 static bool SINC_CheckChannelHPFSaturation (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, refer to sinc_channel_id_t for details.

Return values

<i>true</i>	Selected channel's HPF saturation occurred.
<i>false</i>	Selected channel's HPF saturation did not occurred.

38.5.38 static bool SINC_CheckChannelShiftSaturation (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, refer to sinc_channel_id_t for details.

Return values

<i>true</i>	Selected channel's shift saturation occurred.
<i>false</i>	Selected channel's shift saturation did not occurred.

38.5.39 static bool SINC_CheckChannelBiasSaturation (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, refer to sinc_channel_id_t for details.

Return values

<i>true</i>	Selected channel's bias saturation occurred.
<i>false</i>	Selected channel's bias saturation did not occurred.

38.5.40 uint8_t SINC_GetChannelConversionCount (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*)

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, refer to sinc_channel_id_t for details.

Returns

uint8_t Selected channel's number of conversions.

38.5.41 static bool SINC_CheckChannelConvProgress (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	The id of sinc channel to check.

Return values

<i>false</i>	Selected channel conversion not in progress.
<i>true</i>	Selected channel conversion in progress.

38.5.42 static bool SINC_CheckChannelReadyForConv (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	The id of sinc channel to check.

Return values

<i>true</i>	Selected channel is ready for conversion.
<i>false</i>	Selected channel is not ready for conversion.

38.5.43 static void SINC_SetChannelLowLimitThreshold (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **uint32_t** *lowLimitThreshold*) [inline], [static]

Note

When the data exceeds the low-limit threshold value, a low-limit event occurs, and the limit threshold format is determined by channel's result data format [sinc_result_data_format_t](#).

Low limit value must be lower than high limit value, otherwise the low-limit threshold does not work.

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>lowLimit-Threshold</i>	Specify the low-limit threshold value, range from 0 to 0xFFFFFUL.

**38.5.44 static void SINC_SetChannelHighLimitThreshold (SINC_Type * *base*,
 sinc_channel_id_t *chId*, uint32_t *highLimitThreshold*) [inline],
 [static]**

Note

When the data exceeds the high-limit threshold value, a high-limit event occurs, and the limit threshold format is determined by channel's result data format [sinc_result_data_format_t](#).

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>highLimitThreshold</i>	Specify the high-limit threshold value, range from 0 to 0xFFFFFFFUL.

**38.5.45 static void SINC_SetChannelLimitDetectorMode (SINC_Type * *base*,
 sinc_channel_id_t *chId*, sinc_limit_detector_mode_t *mode*) [inline],
 [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>mode</i>	Specify the mode of limit detector, please refer to sinc_limit_detector_mode_t .

**38.5.46 static void SINC_EnableChannelHighLimitBreakSignal (SINC_Type * *base*,
 sinc_channel_id_t *chId*, bool *enable*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable high limit break signal: <ul style="list-style-type: none"> • true Enable the automatic assertion of the BREAK_HIGH signal when SINC detects a high-limit event on the selected channel. • false Disable high limit break signal.

38.5.47 **static void SINC_EnableChannelWindowLimitBreakSignal (SINC_Type *
base, sinc_channel_id_t chId, bool enable) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable window limit break signal: <ul style="list-style-type: none">• true Enable the automatic assertion of the BREAK_WIN signal when SINC detects a window-limit event on the selected channel.• false Disable window limit break signal.

38.5.48 static void SINC_EnableChannelLowLimitBreakSignal (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable low limit break signal: <ul style="list-style-type: none">• true Enable the automatic assertion of the BREAK_LOW signal when SINC detects a low-limit event on the selected channel.• false Disable low limit break signal.

38.5.49 static void SINC_SetChannelScdOperateMode (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **sinc_scd_operate_mode_t** *opMode*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>opMode</i>	Specify the operate mode to set, please refer to sinc_scd_operate_mode_t .

38.5.50 static void SINC_SetChannelScdLimitThreshold (**SINC_Type** * *base*, **sinc_channel_id_t** *chId*, **uint8_t** *u8ScdLimitThreshold*) [inline], [static]

Note

The SCD counter tracks the number of received bits with the same repeating value(always 0 or always 1, set by [SINC_SetChannelScdOption\(\)](#)), if that number exceeds the scdLimitThreshold, an SCD event occurs on the associated channel.

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>u8ScdLimit-Threshold</i>	Specify the threshold value for the SCD counter, range from 2 to 255.

**38.5.51 static void SINC_SetChannelScdOption (SINC_Type * *base*,
sinc_channel_id_t chId, *sinc_scd_option_t option*) [inline],
[static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>option</i>	Specify which repeating bit value increments the SCD counter.

**38.5.52 static void SINC_EnableChannelScdBreakSignal (SINC_Type * *base*,
sinc_channel_id_t chId, *bool enable*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable SCD break signal: <ul style="list-style-type: none"> • true Enable SCD break signal. • false Disable SCD break signal.

**38.5.53 static void SINC_SetChannelCadLimitThreshold (*SINC_Type* * *base*,
sinc_channel_id_t *chId*, *sinc_cad_threshold_t* *cadLimitThreshold*)
[inline], [static]**

Note

The CAD counter tracks the number of clock cycles during which SINC does not detect a clock, if that number exceeds the threshold value, a CAD event occurs on the selected channel.

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>cadLimitThreshold</i>	Specify the threshold value for the CAD counter, please refer to sinc_cad_threshold_t .

**38.5.54 static void SINC_EnableChannelCadBreakSignal (*SINC_Type* * *base*,
sinc_channel_id_t *chId*, *bool* *enable*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>enable</i>	Used to enable/disable CAD break signal: <ul style="list-style-type: none"> • true Enable selected channel's CAD break signal; • false Disable selected channel's CAD break signal.

**38.5.55 static void SINC_SetChannelZcdOperateMode (*SINC_Type* * *base*,
sinc_channel_id_t *chId*, *sinc_zero_cross_operate_mode_t* *opMode*)
[inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>opMode</i>	Specify the operate mode, please refer to sinc_zero_cross_operate_mode_t .

**38.5.56 static void SINC_SetChannelPulseTriggerMux (SINC_Type * *base*,
 sinc_channel_id_t *chId*, sinc_pulse_trigger_mux_t *pulseTrigMux*)
[inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>pulseTrigMux</i>	Used to selected the signal for pulse-trigger output.

**38.5.57 static void SINC_SetChannelDebugOutput (SINC_Type * *base*,
 sinc_channel_id_t *chId*, sinc_debug_output_t *debugOutput*) [inline],
[static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .
<i>debugOutput</i>	Used to select debug output, please refer to sinc_debug_output_t .

**38.5.58 static void SINC_LatchChannelDebugProcedure (SINC_Type * *base*,
 sinc_channel_id_t *chId*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .

**38.5.59 static bool SINC_CheckChannelDebugDataValid (SINC_Type * *base*,
 sinc_channel_id_t *chId*) [inline], [static]**

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .

Return values

<i>true</i>	Data is valid.
<i>false</i>	Data is invalid.

38.5.60 static uint32_t SINC_GetChannelDebugData (*SINC_Type* * *base*, *sinc_channel_id_t* *chId*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>chId</i>	Selected channel id, please refer to sinc_channel_id_t .

Returns

Selected channel's debug data.

38.5.61 static void SINC_EnableInterrupts (*SINC_Type* * *base*, *uint64_t* *interruptMasks*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>interruptMasks</i>	Mask of interrupts to enable, should be the OR'ed value of sinc_interrupt_enable_t .

38.5.62 static void SINC_DisableInterrupts (*SINC_Type* * *base*, *uint64_t* *interruptMasks*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>interruptMasks</i>	Mask of interrupts to disable, should be the OR'ed value of sinc_interrupt_enable_t .

38.5.63 static uint64_t SINC_GetInterruptStatus (**SINC_Type** * *base*) [inline], [static]

Parameters

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

Returns

SINC module's interrupt status flags, the OR'ed value of [sinc_interrupt_status_t](#).

38.5.64 static void SINC_ClearInterruptStatus (**SINC_Type** * *base*, **uint64_t** *statusMasks*) [inline], [static]

Parameters

<i>base</i>	SINC peripheral base address.
<i>statusMasks</i>	The mask of interrupt status flags to clear, should be the OR'ed value of sinc_interrupt_status_t .

Chapter 39

TEMPSENSOR: Temperature Sensor Module

39.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Temperature Sensor Module (TEMPSENSOR) module of MCUXpresso SDK devices.

39.2 TEMPSENSOR Operations

The function `TEMPMON_Init()` will initialize the TEMPSENSOR peripheral operation.

The function `TEMPMON_Deinit()` will deinitialize the TEMPSENSOR peripheral operation.

The function `TEMPMON_GetDefaultConfig()` will get default configuration.

The function `TEMPMON_StartMeasure()` will start the temperature measurement process.

The function `TEMPMON_StopMeasure()` will stop the temperature measurement process.

The function `TEMPMON_GetCurrentTemp()` will get the current temperature.

The function `TEMPMON_SetTempAlarm()` will set the temperature count that will generate an alarm interrupt.

Files

- file `fsl_tempsensor.h`

Data Structures

- struct `tmpsns_config_t`
TMPNSN temperature structure. [More...](#)

Enumerations

- enum {
 `kTEMPSENSOR_HighTempInterruptStatusEnable`,
 `kTEMPSENSOR_LowTempInterruptStatusEnable`,
 `kTEMPSENSOR_PanicTempInterruptStatusEnable`,
 `kTEMPSENSOR_FinishInterruptStatusEnable` = `TMPNSN_CTRL1_FINISH_IE_MASK` }
TMPNSN interrupt status enable type, `tmpsns_interrupt_status_enable_t`.
- enum {
 `kTEMPSENSOR_HighTempInterruptStatus` = `TMPNSN_STATUS0_HIGH_TEMP_MASK`,
 `kTEMPSENSOR_LowTempInterruptStatus` = `TMPNSN_STATUS0_LOW_TEMP_MASK`,
 `kTEMPSENSOR_PanicTempInterruptStatus` = `TMPNSN_STATUS0_PANIC_TEMP_MASK` }
TMPNSN interrupt status type, `tmpsns_interrupt_status_t`.

- enum `tmpsns_measure_mode_t` {

 `kTEMPSENSOR_SingleMode` = 0U,

 `kTEMPSENSOR_ContinuousMode` = 1U }

 TMPSENSE measure mode, tempsensor_measure_mode.
- enum `tmpsns_alarm_mode_t` {

 `kTEMPMON_HighAlarmMode` = 0U,

 `kTEMPMON_PanicAlarmMode` = 1U,

 `kTEMPMON_LowAlarmMode` = 2U }

 TMPSENSE alarm mode.

Functions

- void `TMPSENSE_Init` (`TMPSENSE_Type` *base, const `tmpsns_config_t` *config)

 Initializes the TMPSENSE module.
- void `TMPSENSE_Deinit` (`TMPSENSE_Type` *base)

 Deinitializes the TMPSENSE module.
- void `TMPSENSE_GetDefaultConfig` (`tmpsns_config_t` *config)

 Gets the default configuration structure.
- void `TMPSENSE_StartMeasure` (`TMPSENSE_Type` *base)

 start the temperature measurement process.
- void `TMPSENSE_StopMeasure` (`TMPSENSE_Type` *base)

 stop the measurement process.
- float `TMPSENSE_GetCurrentTemperature` (`TMPSENSE_Type` *base)

 Get current temperature with the fused temperature calibration data.
- void `TMPSENSE_SetTempAlarm` (`TMPSENSE_Type` *base, `int32_t` tempVal, `tmpsns_alarm_mode_t` alarmMode)

 Set the temperature count (raw sensor output) that will generate an alarm interrupt.
- void `TMPSENSE_EnableInterrupt` (`TMPSENSE_Type` *base, `uint32_t` mask)

 Enable interrupt status.
- void `TMPSENSE_DisableInterrupt` (`TMPSENSE_Type` *base, `uint32_t` mask)

 Disable interrupt status.
- static `uint32_t` `TMPSENSE_GetInterruptFlags` (`TMPSENSE_Type` *base)

 Get interrupt status flag.
- static void `TMPSENSE_ClearInterruptFlags` (`TMPSENSE_Type` *base, `uint32_t` mask)

 Clear interrupt status flag.

Driver version

- #define `FSL_TMPSENSE_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 2))

39.3 Data Structure Documentation

39.3.1 struct `tmpsns_config_t`

Data Fields

- `tmpsns_measure_mode_t measureMode`

 The temperature measure mode.
- `uint16_t frequency`

- `int32_t highAlarmTemp`
The temperature measure frequency.
- `int32_t panicAlarmTemp`
The high alarm temperature.
- `int32_t lowAlarmTemp`
The panic alarm temperature.
- `int32_t lowAlarmTemp`
The low alarm temperature.

Field Documentation

- (1) `tmpsns_measure_mode_t tmpsns_config_t::measureMode`
- (2) `uint16_t tmpsns_config_t::frequency`
- (3) `int32_t tmpsns_config_t::highAlarmTemp`
- (4) `int32_t tmpsns_config_t::panicAlarmTemp`
- (5) `int32_t tmpsns_config_t::lowAlarmTemp`

39.4 Enumeration Type Documentation

39.4.1 anonymous enum

Enumerator

- `kTEMPSENSOR_HighTempInterruptStatusEnable` High temperature interrupt status enable.
`kTEMPSENSOR_LowTempInterruptStatusEnable` Low temperature interrupt status enable.
`kTEMPSENSOR_PanicTempInterruptStatusEnable` Panic temperature interrupt status enable.
`kTEMPSENSOR_FinishInterruptStatusEnable` Finish interrupt enable.

39.4.2 anonymous enum

Enumerator

- `kTEMPSENSOR_HighTempInterruptStatus` High temperature interrupt status.
`kTEMPSENSOR_LowTempInterruptStatus` Low temperature interrupt status.
`kTEMPSENSOR_PanicTempInterruptStatus` Panic temperature interrupt status.

39.4.3 enum tmpsns_measure_mode_t

Enumerator

- `kTEMPSENSOR_SingleMode` Single measurement mode.
`kTEMPSENSOR_ContinuousMode` Continuous measurement mode.

39.4.4 enum tmpsns_alarm_mode_t

Enumerator

kTEMPMON_HighAlarmMode The high alarm temperature interrupt mode.

kTEMPMON_PanicAlarmMode The panic alarm temperature interrupt mode.

kTEMPMON_LowAlarmMode The low alarm temperature interrupt mode.

39.5 Function Documentation

39.5.1 void TMPSNS_Init (**TMPSNS_Type** * *base*, **const tmpsns_config_t** * *config*)

Parameters

<i>base</i>	TMPSNS base pointer
<i>config</i>	Pointer to configuration structure.

39.5.2 void TMPSNS_Deinit (**TMPSNS_Type** * *base*)

Parameters

<i>base</i>	TMPSNS base pointer
-------------	---------------------

39.5.3 void TMPSNS_GetDefaultConfig (**tmpsns_config_t** * *config*)

This function initializes the TMPSNS configuration structure to a default value. The default values are: tempmonConfig->frequency = 0x02U; tempmonConfig->highAlarmTemp = 44U; tempmonConfig->panicAlarmTemp = 90U; tempmonConfig->lowAlarmTemp = 39U;

Parameters

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

39.5.4 void TMPSNS_StartMeasure (**TMPSNS_Type** * *base*)

Parameters

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

39.5.5 void TMPSEN_StopMeasure (**TMPSEN_Type * base**)

Parameters

<i>base</i>	TMPSEN base pointer
-------------	---------------------

39.5.6 float TMPSEN_GetCurrentTemperature (**TMPSEN_Type * base**)

Parameters

<i>base</i>	TMPSEN base pointer
-------------	---------------------

Returns

current temperature with degrees Celsius.

39.5.7 void TMPSEN_SetTempAlarm (**TMPSEN_Type * base, int32_t tempVal, tmpsns_alarm_mode_t alarmMode**)

Parameters

<i>base</i>	TMPSEN base pointer
<i>tempVal</i>	The alarm temperature with degrees Celsius
<i>alarmMode</i>	The alarm mode.

39.5.8 void TMPSEN_EnableInterrupt (**TMPSEN_Type * base, uint32_t mask**)

Parameters

<i>base</i>	TMPNS base pointer
<i>mask</i>	The interrupts to enable from tmpsns_interrupt_status_enable_t.

39.5.9 void TMPNS_DisableInterrupt (**TMPNS_Type * base, uint32_t mask**)

Parameters

<i>base</i>	TMPNS base pointer
<i>mask</i>	The interrupts to disable from tmpsns_interrupt_status_enable_t.

39.5.10 static uint32_t TMPNS_GetInterruptFlags (**TMPNS_Type * base**) [inline], [static]

Parameters

<i>base</i>	TMPNS base pointer
-------------	--------------------

39.5.11 static void TMPNS_ClearInterruptFlags (**TMPNS_Type * base, uint32_t mask**) [inline], [static]

Parameters

<i>base</i>	TMPNS base pointer
<i>mask</i>	The interrupts to disable from tmpsns_interrupt_status_t.

Chapter 40

TPM: Timer PWM Module

40.1 Overview

The MCUXpresso SDK provides a driver for the Timer PWM Module (TPM) of MCUXpresso SDK devices.

The TPM driver supports the generation of PWM signals, input capture, and output compare modes. On some SoCs, the driver supports the generation of combined PWM signals, dual-edge capture, and quadrature decoder modes. The driver also supports configuring each of the TPM fault inputs. The fault input is available only on some SoCs.

40.2 Introduction of TPM

40.2.1 Initialization and deinitialization

The function [TPM_Init\(\)](#) initializes the TPM with a specified configurations. The function [TPM_GetDefaultConfig\(\)](#) gets the default configurations. On some SoCs, the initialization function issues a software reset to reset the TPM internal logic. The initialization function configures the TPM's behavior when it receives a trigger input and its operation in doze and debug modes.

The function [TPM_Deinit\(\)](#) disables the TPM counter and turns off the module clock.

40.2.2 PWM Operations

The function [TPM_SetupPwm\(\)](#) sets up TPM channels for the PWM output. The function can set up the PWM signal properties for multiple channels. Each channel has its own [tpm_chnl_pwm_signal_param_t](#) structure that is used to specify the output signals duty cycle and level-mode. However, the same PWM period and PWM mode is applied to all channels requesting a PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 where 0=inactive signal (0% duty cycle) and 100=always active signal (100% duty cycle). When generating a combined PWM signal, the channel number passed refers to a channel pair number, for example 0 refers to channel 0 and 1, 1 refers to channels 2 and 3.

The function [TPM_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular TPM channel.

The function [TPM_UpdateChnlEdgeLevelSelect\(\)](#) updates the level select bits of a particular TPM channel. This can be used to disable the PWM output when making changes to the PWM signal.

40.2.3 Input capture operations

The function [TPM_SetupInputCapture\(\)](#) sets up a TPM channel for input capture. The user can specify the capture edge.

The function [TPM_SetupDualEdgeCapture\(\)](#) can be used to measure the pulse width of a signal. This is available only for certain SoCs. A channel pair is used during the capture with the input signal coming through a channel that can be configured. The user can specify the capture edge for each channel and any filter value to be used when processing the input signal.

40.2.4 Output compare operations

The function [TPM_SetupOutputCompare\(\)](#) sets up a TPM channel for output comparison. The user can specify the channel output on a successful comparison and a comparison value.

40.2.5 Quad decode

The function [TPM_SetupQuadDecode\(\)](#) sets up TPM channels 0 and 1 for quad decode, which is available only for certain SoCs. The user can specify the quad decode mode, polarity, and filter properties for each input signal.

40.2.6 Fault operation

The function [TPM_SetupFault\(\)](#) sets up the properties for each fault, which is available only for certain SoCs. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

40.2.7 Status

Provides functions to get and clear the TPM status.

40.2.8 Interrupt

Provides functions to enable/disable TPM interrupts and get current enabled interrupts.

40.3 Typical use case

40.3.1 PWM output

Output the PWM signal on 2 TPM channels with different duty cycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/tpm

Data Structures

- struct `tpm_chnl_pwm_signal_param_t`
Options to configure a TPM channel's PWM signal. [More...](#)
- struct `tpm_dual_edge_capture_param_t`
TPM dual edge capture parameters. [More...](#)
- struct `tpm_phase_params_t`
TPM quadrature decode phase parameters. [More...](#)
- struct `tpm_config_t`
TPM config structure. [More...](#)

Macros

- #define `TPM_MAX_COUNTER_VALUE(x)` ((1U != (uint8_t)FSL_FEATURE_TPM_HAS_32BIT_COUNTERn(x)) ? 0xFFFFU : 0xFFFFFFFFU)
Help macro to get the max counter value.

Enumerations

- enum `tpm_chnl_t` {

 `kTPM_Chnl_0` = 0U,

 `kTPM_Chnl_1`,

 `kTPM_Chnl_2`,

 `kTPM_Chnl_3`,

 `kTPM_Chnl_4`,

 `kTPM_Chnl_5`,

 `kTPM_Chnl_6`,

 `kTPM_Chnl_7` }

List of TPM channels.
- enum `tpm_pwm_mode_t` {

 `kTPM_EdgeAlignedPwm` = 0U,

 `kTPM_CenterAlignedPwm`,

 `kTPM_CombinedPwm` }

TPM PWM operation modes.
- enum `tpm_pwm_level_select_t` {

 `kTPM_HighTrue` = 0U,

 `kTPM_LowTrue` }

TPM PWM output pulse mode: high-true, low-true or no output.
- enum `tpm_pwm_pause_level_select_t` {

 `kTPM_ClearOnPause` = 0U,

 `kTPM_SetOnPause` }

- *TPM PWM output when first enabled or paused: set or clear.*
- enum `tpm_chnl_control_bit_mask_t` {

 kTPM_ChnlELSnAMask = TPM_CnSC_ELSA_MASK,
 kTPM_ChnlELSnBMask = TPM_CnSC_ELSB_MASK,
 kTPM_ChnlMSAMask = TPM_CnSC_MSA_MASK,
 kTPM_ChnlMSBMask = TPM_CnSC_MSB_MASK }

List of TPM channel modes and level control bit mask.
- enum `tpm_trigger_select_t`
Trigger sources available.
- enum `tpm_trigger_source_t` {

 kTPM_TriggerSource_External = 0U,
 kTPM_TriggerSource_Internal }

Trigger source options available.
- enum `tpm_ext_trigger_polarity_t` {

 kTPM_ExtTrigger_Active_High = 0U,
 kTPM_ExtTrigger_Active_Low }

External trigger source polarity.
- enum `tpm_output_compare_mode_t` {

 kTPM_NoOutputSignal = (1U << TPM_CnSC_MSA_SHIFT),
 kTPM_ToggleOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
 kTPM_ClearOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)),
 kTPM_SetOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (3U << TPM_CnSC_ELSA_SHIFT)),
 kTPM_HighPulseOutput = ((3U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
 kTPM_LowPulseOutput = ((3U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)) }

TPM output compare modes.
- enum `tpm_input_capture_edge_t` {

 kTPM_RisingEdge = (1U << TPM_CnSC_ELSA_SHIFT),
 kTPM_FallingEdge = (2U << TPM_CnSC_ELSA_SHIFT),
 kTPM_RiseAndFallEdge = (3U << TPM_CnSC_ELSA_SHIFT) }

TPM input capture edge.
- enum `tpm_quad_decode_mode_t` {

 kTPM_QuadPhaseEncode = 0U,
 kTPM_QuadCountAndDir }

TPM quadrature decode modes.
- enum `tpm_phase_polarity_t` {

 kTPM_QuadPhaseNormal = 0U,
 kTPM_QuadPhaseInvert }

TPM quadrature phase polarities.
- enum `tpm_clock_source_t` {

 kTPM_SystemClock = 1U,
 kTPM_ExternalClock,
 kTPM_ExternalInputTriggerClock }

- *TPM clock source selection.*
- enum `tpm_clock_prescale_t` {

kTPM_Prescale_Divide_1 = 0U,

kTPM_Prescale_Divide_2,

kTPM_Prescale_Divide_4,

kTPM_Prescale_Divide_8,

kTPM_Prescale_Divide_16,

kTPM_Prescale_Divide_32,

kTPM_Prescale_Divide_64,

kTPM_Prescale_Divide_128 }
- *TPM prescale value selection for the clock source.*
- enum `tpm_interrupt_enable_t` {

kTPM_Chnl0InterruptEnable = (1U << 0),

kTPM_Chnl1InterruptEnable = (1U << 1),

kTPM_Chnl2InterruptEnable = (1U << 2),

kTPM_Chnl3InterruptEnable = (1U << 3),

kTPM_Chnl4InterruptEnable = (1U << 4),

kTPM_Chnl5InterruptEnable = (1U << 5),

kTPM_Chnl6InterruptEnable = (1U << 6),

kTPM_Chnl7InterruptEnable = (1U << 7),

kTPM_TimeOverflowInterruptEnable = (1U << 8) }
- *List of TPM interrupts.*
- enum `tpm_status_flags_t` {

kTPM_Chnl0Flag = (1U << 0),

kTPM_Chnl1Flag = (1U << 1),

kTPM_Chnl2Flag = (1U << 2),

kTPM_Chnl3Flag = (1U << 3),

kTPM_Chnl4Flag = (1U << 4),

kTPM_Chnl5Flag = (1U << 5),

kTPM_Chnl6Flag = (1U << 6),

kTPM_Chnl7Flag = (1U << 7),

kTPM_TimeOverflowFlag = (1U << 8) }
- *List of TPM flags.*

Functions

- static void `TPM_Reset` (TPM_Type *base)
Performs a software reset on the TPM module.

Driver version

- #define `FSL TPM_DRIVER_VERSION` (MAKE_VERSION(2, 2, 3))
TPM driver version 2.2.3.

Initialization and deinitialization

- void `TPM_Init` (TPM_Type *base, const `tpm_config_t` *config)

Ungates the TPM clock and configures the peripheral for basic operation.

- void **TPM_Deinit** (TPM_Type *base)

Stops the counter and gates the TPM clock.
- void **TPM_GetDefaultConfig** (tpm_config_t *config)

Fill in the TPM config struct with the default settings.
- tpm_clock_prescale_t **TPM_CalculateCounterClkDiv** (TPM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

Calculates the counter clock prescaler.

Channel mode operations

- status_t **TPM_SetupPwm** (TPM_Type *base, const tpm_chnl_pwm_signal_param_t *chnlParams, uint8_t numOfChnls, tpm_pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.
- status_t **TPM_UpdatePwmDutycycle** (TPM_Type *base, tpm_chnl_t chnlNumber, tpm_pwm_mode_t currentPwmMode, uint8_t dutyCyclePercent)

Update the duty cycle of an active PWM signal.
- void **TPM_UpdateChnlEdgeLevelSelect** (TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t level)

Update the edge level selection for a channel.
- static uint8_t **TPM_GetChannelContorlBits** (TPM_Type *base, tpm_chnl_t chnlNumber)

Get the channel control bits value (mode, edge and level bit fileds).
- static void **TPM_DisableChannel** (TPM_Type *base, tpm_chnl_t chnlNumber)

Dsiable the channel.
- static void **TPM_EnableChannel** (TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t control)

Enable the channel according to mode and level configs.
- void **TPM_SetupInputCapture** (TPM_Type *base, tpm_chnl_t chnlNumber, tpm_input_capture_edge_t captureMode)

Enables capturing an input signal on the channel using the function parameters.
- void **TPM_SetupOutputCompare** (TPM_Type *base, tpm_chnl_t chnlNumber, tpm_output_compare_mode_t compareMode, uint32_t compareValue)

Configures the TPM to generate timed pulses.
- void **TPM_SetupDualEdgeCapture** (TPM_Type *base, tpm_chnl_t chnlPairNumber, const tpm_dual_edge_capture_param_t *edgeParam, uint32_t filterValue)

Configures the dual edge capture mode of the TPM.
- void **TPM_SetupQuadDecode** (TPM_Type *base, const tpm_phase_params_t *phaseAParams, const tpm_phase_params_t *phaseBParams, tpm_quad_decode_mode_t quadMode)

Configures the parameters and activates the quadrature decode mode.
- static void **TPM_SetChannelPolarity** (TPM_Type *base, tpm_chnl_t chnlNumber, bool enable)

Set the input and output polarity of each of the channels.
- static void **TPM_EnableChannelExtTrigger** (TPM_Type *base, tpm_chnl_t chnlNumber, bool enable)

Enable external trigger input to be used by channel.

Interrupt Interface

- void **TPM_EnableInterrupts** (TPM_Type *base, uint32_t mask)

Enables the selected TPM interrupts.
- void **TPM_DisableInterrupts** (TPM_Type *base, uint32_t mask)

Disables the selected TPM interrupts.
- uint32_t **TPM_GetEnabledInterrupts** (TPM_Type *base)

Gets the enabled TPM interrupts.

Status Interface

- static uint32_t [TPM_GetChannelValue](#) (TPM_Type *base, [tpm_chnl_t](#) chnlNumber)
Gets the TPM channel value.
- static uint32_t [TPM_GetStatusFlags](#) (TPM_Type *base)
Gets the TPM status flags.
- static void [TPM_ClearStatusFlags](#) (TPM_Type *base, uint32_t mask)
Clears the TPM status flags.

Read and write the timer period

- static void [TPM_SetTimerPeriod](#) (TPM_Type *base, uint32_t ticks)
Sets the timer period in units of ticks.
- static uint32_t [TPM_GetCurrentTimerCount](#) (TPM_Type *base)
Reads the current timer counting value.

Timer Start and Stop

- static void [TPM_StartTimer](#) (TPM_Type *base, [tpm_clock_source_t](#) clockSource)
Starts the TPM counter.
- static void [TPM_StopTimer](#) (TPM_Type *base)
Stops the TPM counter.

40.4 Data Structure Documentation

40.4.1 struct tpm_chnl_pwm_signal_param_t

Data Fields

- [tpm_chnl_t](#) chnlNumber
TPM channel to configure.
- [tpm_pwm_pause_level_select_t](#) pauseLevel
PWM output level when counter first enabled or paused.
- [tpm_pwm_level_select_t](#) level
PWM output active level select.
- uint8_t [dutyCyclePercent](#)
PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...
- uint8_t [firstEdgeDelayPercent](#)
Used only in combined PWM mode to generate asymmetrical PWM.
- bool [enableComplementary](#)
Used only in combined PWM mode.
- [tpm_pwm_pause_level_select_t](#) secPauseLevel
Used only in combined PWM mode.
- uint8_t [deadTimeValue](#) [2]
The dead time value for channel n and n+1 in combined complementary PWM mode.

Field Documentation

(1) **tpm_chnl_t tpm_chnl_pwm_signal_param_t::chnlNumber**

In combined mode (available in some SoC's), this represents the channel pair number

(2) **uint8_t tpm_chnl_pwm_signal_param_t::dutyCyclePercent**

100=always active signal (100% duty cycle)

(3) **uint8_t tpm_chnl_pwm_signal_param_t::firstEdgeDelayPercent**

Specifies the delay to the first edge in a PWM period. If unsure, leave as 0. Should be specified as percentage of the PWM period, (dutyCyclePercent + firstEdgeDelayPercent) value should be not greater than 100.

(4) **bool tpm_chnl_pwm_signal_param_t::enableComplementary**

true: The combined channels output complementary signals; false: The combined channels output same signals;

(5) **tpm_pwm_pause_level_select_t tpm_chnl_pwm_signal_param_t::secPauseLevel**

Define the second channel output level when counter first enabled or paused

(6) **uint8_t tpm_chnl_pwm_signal_param_t::deadTimeValue[2]**

Deadtime insertion is disabled when this value is zero, otherwise deadtime insertion for channel n/n+1 is configured as (deadTimeValue * 4) clock cycles. deadTimeValue's available range is 0 ~ 15.

40.4.2 struct tpm_dual_edge_capture_param_t

Note

This mode is available only on some SoC's.

Data Fields

- bool **enableSwap**
true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored
- **tpm_input_capture_edge_t currChanEdgeMode**
Input capture edge select for channel n.
- **tpm_input_capture_edge_t nextChanEdgeMode**
Input capture edge select for channel n+1.

40.4.3 struct tpm_phase_params_t

Data Fields

- `uint32_t phaseFilterVal`
Filter value, filter is disabled when the value is zero.
- `tpm_phase_polarity_t phasePolarity`
Phase polarity.

40.4.4 struct tpm_config_t

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the `TPM_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Data Fields

- `tpm_clock_prescale_t prescale`
Select TPM clock prescale value.
- `bool useGlobalTimeBase`
true: The TPM channels use an external global time base (the local counter still use for generate overflow interrupt and DMA request); false: All TPM channels use the local counter as their timebase
- `bool syncGlobalTimeBase`
true: The TPM counter is synchronized to the global time base; false: disabled
- `tpm_trigger_select_t triggerSelect`
Input trigger to use for controlling the counter operation.
- `tpm_trigger_source_t triggerSource`
Decides if we use external or internal trigger.
- `tpm_ext_trigger_polarity_t extTriggerPolarity`
when using external trigger source, need selects the polarity of it.
- `bool enableDoze`
true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode
- `bool enableDebugMode`
true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode
- `bool enableReloadOnTrigger`
true: TPM counter is reloaded on trigger; false: TPM counter not reloaded
- `bool enableStopOnOverflow`
true: TPM counter stops after overflow; false: TPM counter continues running after overflow
- `bool enableStartOnTrigger`
true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately
- `bool enablePauseOnTrigger`
true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running
- `uint8_t chnlPolarity`
Defines the input/output polarity of the channels in POL register.

Field Documentation

- (1) `tpm_trigger_source_t tpm_config_t::triggerSource`
- (2) `tpm_ext_trigger_polarity_t tpm_config_t::extTriggerPolarity`

40.5 Macro Definition Documentation

40.5.1 #define FSL TPM DRIVER VERSION (MAKE_VERSION(2, 2, 3))

40.6 Enumeration Type Documentation**40.6.1 enum tpm_chnl_t**

Note

Actual number of available channels is SoC dependent

Enumerator

- kTPM_Chnl_0* TPM channel number 0.
- kTPM_Chnl_1* TPM channel number 1.
- kTPM_Chnl_2* TPM channel number 2.
- kTPM_Chnl_3* TPM channel number 3.
- kTPM_Chnl_4* TPM channel number 4.
- kTPM_Chnl_5* TPM channel number 5.
- kTPM_Chnl_6* TPM channel number 6.
- kTPM_Chnl_7* TPM channel number 7.

40.6.2 enum tpm_pwm_mode_t

Enumerator

- kTPM_EdgeAlignedPwm* Edge aligned PWM.
- kTPM_CenterAlignedPwm* Center aligned PWM.
- kTPM_CombinedPwm* Combined PWM (Edge-aligned, center-aligned, or asymmetrical PWMs can be obtained in combined mode using different software configurations)

40.6.3 enum tpm_pwm_level_select_t

Note

When the TPM has PWM pause level select feature, the PWM output cannot be turned off by selecting the output level. In this case, the channel must be closed to close the PWM output.

Enumerator

kTPM_HighTrue High true pulses.

kTPM_LowTrue Low true pulses.

40.6.4 enum tpm_pwm_pause_level_select_t

Enumerator

kTPM_ClearOnPause Clear Output when counter first enabled or paused.

kTPM_SetOnPause Set Output when counter first enabled or paused.

40.6.5 enum tpm_chnl_control_bit_mask_t

Enumerator

kTPM_ChnlELSnAMask Channel ELSA bit mask.

kTPM_ChnlELSnBMask Channel ELSB bit mask.

kTPM_ChnlMSAMask Channel MSA bit mask.

kTPM_ChnlMSBMask Channel MSB bit mask.

40.6.6 enum tpm_trigger_select_t

This is used for both internal & external trigger sources (external trigger sources available in certain SoC's)

Note

The actual trigger sources available is SoC-specific.

40.6.7 enum tpm_trigger_source_t

Note

This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

Enumerator

kTPM_TriggerSource_External Use external trigger input.

kTPM_TriggerSource_Internal Use internal trigger (channel pin input capture)

40.6.8 enum tpm_ext_trigger_polarity_t

Note

Selects the polarity of the external trigger source.

Enumerator

kTPM_ExtTrigger_Active_High External trigger input is active high.

kTPM_ExtTrigger_Active_Low External trigger input is active low.

40.6.9 enum tpm_output_compare_mode_t

Enumerator

kTPM_NoOutputSignal No channel output when counter reaches CnV.

kTPM_ToggleOnMatch Toggle output.

kTPM_ClearOnMatch Clear output.

kTPM_SetOnMatch Set output.

kTPM_HighPulseOutput Pulse output high.

kTPM_LowPulseOutput Pulse output low.

40.6.10 enum tpm_input_capture_edge_t

Enumerator

kTPM_RisingEdge Capture on rising edge only.

kTPM_FallingEdge Capture on falling edge only.

kTPM_RiseAndFallEdge Capture on rising or falling edge.

40.6.11 enum tpm_quad_decode_mode_t

Note

This mode is available only on some SoC's.

Enumerator

kTPM_QuadPhaseEncode Phase A and Phase B encoding mode.

kTPM_QuadCountAndDir Count and direction encoding mode.

40.6.12 enum tpm_phase_polarity_t

Enumerator

kTPM_QuadPhaseNormal Phase input signal is not inverted.

kTPM_QuadPhaseInvert Phase input signal is inverted.

40.6.13 enum tpm_clock_source_t

Enumerator

kTPM_SystemClock System clock.

kTPM_ExternalClock External TPM_EXTCLK pin clock.

kTPM_ExternalInputTriggerClock Selected external input trigger clock.

40.6.14 enum tpm_clock_prescale_t

Enumerator

kTPM_Prescale_Divide_1 Divide by 1.

kTPM_Prescale_Divide_2 Divide by 2.

kTPM_Prescale_Divide_4 Divide by 4.

kTPM_Prescale_Divide_8 Divide by 8.

kTPM_Prescale_Divide_16 Divide by 16.

kTPM_Prescale_Divide_32 Divide by 32.

kTPM_Prescale_Divide_64 Divide by 64.

kTPM_Prescale_Divide_128 Divide by 128.

40.6.15 enum tpm_interrupt_enable_t

Enumerator

kTPM_Chnl0InterruptEnable Channel 0 interrupt.
kTPM_Chnl1InterruptEnable Channel 1 interrupt.
kTPM_Chnl2InterruptEnable Channel 2 interrupt.
kTPM_Chnl3InterruptEnable Channel 3 interrupt.
kTPM_Chnl4InterruptEnable Channel 4 interrupt.
kTPM_Chnl5InterruptEnable Channel 5 interrupt.
kTPM_Chnl6InterruptEnable Channel 6 interrupt.
kTPM_Chnl7InterruptEnable Channel 7 interrupt.
kTPM_TimeOverflowInterruptEnable Time overflow interrupt.

40.6.16 enum tpm_status_flags_t

Enumerator

kTPM_Chnl0Flag Channel 0 flag.
kTPM_Chnl1Flag Channel 1 flag.
kTPM_Chnl2Flag Channel 2 flag.
kTPM_Chnl3Flag Channel 3 flag.
kTPM_Chnl4Flag Channel 4 flag.
kTPM_Chnl5Flag Channel 5 flag.
kTPM_Chnl6Flag Channel 6 flag.
kTPM_Chnl7Flag Channel 7 flag.
kTPM_TimeOverflowFlag Time overflow flag.

40.7 Function Documentation

40.7.1 void TPM_Init (**TPM_Type** * *base*, **const tpm_config_t** * *config*)

Note

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

Parameters

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

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

40.7.2 void TPM_Deinit (**TPM_Type** * *base*)

Parameters

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

40.7.3 void TPM_GetDefaultConfig (**tpm_config_t** * *config*)

The default values are:

```
*     config->prescale = kTPM_Prescale_Divide_1;
*     config->useGlobalTimeBase = false;
*     config->syncGlobalTimeBase = false;
*     config->dozeEnable = false;
*     config->dbgMode = false;
*     config->enableReloadOnTrigger = false;
*     config->enableStopOnOverflow = false;
*     config->enableStartOnTrigger = false;
*#if FSL_FEATURE TPM HAS_PAUSE_COUNTER_ON_TRIGGER
*     config->enablePauseOnTrigger = false;
#endif
*     config->triggerSelect = kTPM_Trigger_Select_0;
*#if FSL_FEATURE TPM HAS_EXTERNAL_TRIGGER_SELECTION
*     config->triggerSource = kTPM_TriggerSource_External;
*     config->extTriggerPolarity = kTPM_ExtTrigger_Active_High;
#endif
*#if defined(FSL_FEATURE TPM HAS_POL) && FSL_FEATURE TPM HAS_POL
*     config->chnlPolarity = 0U;
#endif
*
```

Parameters

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

40.7.4 **tpm_clock_prescale_t** TPM_CalculateCounterClkDiv (**TPM_Type** * *base*, **uint32_t** *counterPeriod_Hz*, **uint32_t** *srcClock_Hz*)

This function calculates the values for SC[PS].

Parameters

<i>base</i>	TPM peripheral base address
<i>counterPeriod-Hz</i>	The desired frequency in Hz which corresponds to the time when the counter reaches the mod value
<i>srcClock_Hz</i>	TPM counter clock in Hz

return Calculated clock prescaler value.

40.7.5 status_t TPM_SetupPwm (**TPM_Type * base**, **const tpm_chnl_pwm_signal_param_t * chnlParams**, **uint8_t numOfChnls**, **tpm_pwm_mode_t mode**, **uint32_t pwmFreq_Hz**, **uint32_t srcClock_Hz**)

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlParams</i>	Array of PWM channel parameters to configure the channel(s)
<i>numOfChnls</i>	Number of channels to configure, this should be the size of the array passed in
<i>mode</i>	PWM operation mode, options available in enumeration tpm_pwm_mode_t
<i>pwmFreq_Hz</i>	PWM signal frequency in Hz
<i>srcClock_Hz</i>	TPM counter clock in Hz

Returns

kStatus_Success if the PWM setup was successful, kStatus_Error on failure

40.7.6 status_t TPM_UpdatePwmDutycycle (**TPM_Type * base**, **tpm_chnl_t chnlNumber**, **tpm_pwm_mode_t currentPwmMode**, **uint8_t dutyCyclePercent**)

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number. In combined mode, this represents the channel pair number
<i>currentPwm-Mode</i>	The current PWM mode set during PWM setup
<i>dutyCycle-Percent</i>	New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns

kStatus_Success if the PWM setup was successful, kStatus_Error on failure

40.7.7 void TPM_UpdateChnlEdgeLevelSelect (**TPM_Type** * *base*, **tpm_chnl_t chnlNumber**, **uint8_t level**)

Note

When the TPM has PWM pause level select feature (FSL_FEATURE TPM_HAS_PAUSE_LEVEL_SELECT = 1), the PWM output cannot be turned off by selecting the output level. In this case, must use TPM_DisableChannel API to close the PWM output.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number
<i>level</i>	The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

40.7.8 static uint8_t TPM_GetChannelContorlBits (**TPM_Type** * *base*, **tpm_chnl_t chnlNumber**) [inline], [static]

This function disable the channel by clear all mode and level control bits.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number

Returns

The control bits value. This is the logical OR of members of the enumeration [tpm_chnl_control_bit_mask_t](#).

40.7.9 static void TPM_DisableChannel (**TPM_Type** * *base*, **tpm_chnl_t chnlNumber**) [inline], [static]

This function disable the channel by clear all mode and level control bits.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number

40.7.10 static void TPM_EnableChannel (**TPM_Type** * *base*, **tpm_chnl_t chnlNumber, uint8_t control**) [inline], [static]

This function enable the channel output according to input mode/level config parameters.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number
<i>control</i>	The control bits value. This is the logical OR of members of the enumeration tpm_chnl_control_bit_mask_t .

40.7.11 void TPM_SetupInputCapture (**TPM_Type** * *base*, **tpm_chnl_t chnlNumber, tpm_input_capture_edge_t captureMode**)

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number
<i>captureMode</i>	Specifies which edge to capture

40.7.12 void TPM_SetupOutputCompare (**TPM_Type * *base*, **tpm_chnl_t** *chnlNumber*, **tpm_output_compare_mode_t** *compareMode*, **uint32_t** *compareValue*)**

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number
<i>compareMode</i>	Action to take on the channel output when the compare condition is met
<i>compareValue</i>	Value to be programmed in the CnV register.

40.7.13 void TPM_SetupDualEdgeCapture (**TPM_Type * *base*, **tpm_chnl_t** *chnlPairNumber*, **const tpm_dual_edge_capture_param_t** * *edgeParam*, **uint32_t** *filterValue*)**

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlPair-Number</i>	The TPM channel pair number; options are 0, 1, 2, 3
<i>edgeParam</i>	Sets up the dual edge capture function

<i>filterValue</i>	Filter value, specify 0 to disable filter.
--------------------	--

40.7.14 void TPM_SetupQuadDecode (*TPM_Type* * *base*, const *tpm_phase_params_t* * *phaseAParams*, const *tpm_phase_params_t* * *phaseBParams*, *tpm_quad_decode_mode_t* *quadMode*)

Parameters

<i>base</i>	TPM peripheral base address
<i>phaseAParams</i>	Phase A configuration parameters
<i>phaseBParams</i>	Phase B configuration parameters
<i>quadMode</i>	Selects encoding mode used in quadrature decoder mode

40.7.15 static void TPM_SetChannelPolarity (*TPM_Type* * *base*, *tpm_chnl_t chnlNumber*, *bool enable*) [inline], [static]

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number
<i>enable</i>	true: Set the channel polarity to active high; false: Set the channel polarity to active low;

40.7.16 static void TPM_EnableChannelExtTrigger (*TPM_Type* * *base*, *tpm_chnl_t chnlNumber*, *bool enable*) [inline], [static]

In input capture mode, configures the trigger input that is used by the channel to capture the counter value. In output compare or PWM mode, configures the trigger input used to modulate the channel output. When modulating the output, the output is forced to the channel initial value whenever the trigger is not asserted.

Note

No matter how many external trigger sources there are, only input trigger 0 and 1 are used. The even numbered channels share the input trigger 0 and the odd numbered channels share the second input trigger 1.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number
<i>enable</i>	true: Configures trigger input 0 or 1 to be used by channel; false: Trigger input has no effect on the channel

40.7.17 void TPM_EnableInterrupts (TPM_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	TPM peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration tpm_interrupt_enable_t

40.7.18 void TPM_DisableInterrupts (TPM_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	TPM peripheral base address
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration tpm_interrupt_enable_t

40.7.19 uint32_t TPM_GetEnabledInterrupts (TPM_Type * *base*)

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [tpm_interrupt_enable_t](#)

40.7.20 static uint32_t TPM_GetChannelValue (TPM_Type * *base*, tpm_chnl_t *chnlNumber*) [inline], [static]

Note

The TPM channel value contain the captured TPM counter value for the input modes or the match value for the output modes.

Parameters

<i>base</i>	TPM peripheral base address
<i>chnlNumber</i>	The channel number

Returns

The channle CnV regisyer value.

40.7.21 static uint32_t TPM_GetStatusFlags (TPM_Type * *base*) [inline], [static]

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [tpm_status_flags_t](#)

40.7.22 static void TPM_ClearStatusFlags (TPM_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	TPM peripheral base address
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration tpm_status_flags_t

40.7.23 static void TPM_SetTimerPeriod (TPM_Type * *base*, uint32_t *ticks*) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note

1. This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.
2. Call the utility macros provided in the `fsl_common.h` to convert usec or msec to ticks.

Parameters

<i>base</i>	TPM peripheral base address
<i>ticks</i>	A timer period in units of ticks, which should be equal or greater than 1.

**40.7.24 static uint32_t TPM_GetCurrentTimerCount (`TPM_Type * base`)
[inline], [static]**

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

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

Returns

The current counter value in ticks

40.7.25 static void TPM_StartTimer (`TPM_Type * base`, `tpm_clock_source_t clockSource`) [inline], [static]

Parameters

<i>base</i>	TPM peripheral base address
<i>clockSource</i>	TPM clock source; once clock source is set the counter will start running

40.7.26 static void TPM_StopTimer (`TPM_Type * base`) [inline], [static]

Parameters

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

40.7.27 static void TPM_Reset(TPM_Type * *base*) [inline], [static]

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

Note

TPM software reset is available on certain SoC's only

Parameters

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

Chapter 41

TRDC: Trusted Resource Domain Controller

41.1 Overview

The MCUXpresso SDK provides a driver for the Trusted Resource Domain Controller (TRDC) block of MCUXpresso SDK devices.

41.2 TRDC functions

The TRDC module includes four submodules, as follows:

- TRDC_MGR The Manager submodule coordinates all programming model reads and writes.
- TRDC_DAC The Domain Assignment Controller (DAC) handles resource assignments and generation of the domain identifiers.
- TRDC_MBC The Memory Block Checker (MBC) implements access controls for on-chip internal memories and slave peripherals based on a fixed-sized block format.
- TRDC_MRC The Memory Region Checker (MRC) implements the access controls for external off-chip memories and peripherals based on the pre-programmed region descriptor registers

Accordingly, the TRDC driver functions could be grouped as follows:

- TRDC_MGR functions.
- TRDC_DAC functions.
- TRDC_MBC functions.
- TRDC_MRC functions.

Data Structures

- struct `trdc_hardware_config_t`
TRDC hardware configuration. [More...](#)
- struct `trdc_slave_memory_hardware_config_t`
Hardware configuration of the two slave memories within each MBC(memory block checker). [More...](#)
- struct `trdc_processor_domain_assignment_t`
Domain assignment for the processor bus master. [More...](#)
- struct `trdc_non_processor_domain_assignment_t`
Domain assignment for the non-processor bus master. [More...](#)
- struct `trdc_pid_config_t`
Process identifier(PID) configuration for processor cores. [More...](#)
- struct `trdc_idau_config_t`
IDAU(Implementation-Defined Attribution Unit) configuration for TZ-M function control. [More...](#)
- struct `trdc_flw_config_t`
FLW(Flash Logical Window) configuration. [More...](#)
- struct `trdc_domain_error_t`
TRDC domain error definition. [More...](#)
- struct `trdc_memory_access_control_config_t`
Memory access control configuration for MBC/MRC. [More...](#)

- struct `trdc_mrc_region_descriptor_config_t`
The configuration of each region descriptor per domain per MRC instance. [More...](#)
- struct `trdc_mbc_nse_update_config_t`
The configuration of MBC NSE update. [More...](#)
- struct `trdc_mbc_memory_block_config_t`
The configuration of each memory block per domain per MBC instance. [More...](#)

Enumerations

- enum `trdc_did_sel_t` {

kTRDC_DidMda,
 kTRDC_DidInput,
 kTRDC_DidMdaAndInput,
 kTRDC_DidReserved }

TRDC domain ID select method, the register bit TRDC_MDA_W0_0_DFMT0[DIDS], used for domain hit evaluation.
- enum `trdc_secure_attr_t` {

kTRDC_ForceSecure,
 kTRDC_ForceNonSecure,
 kTRDC_MasterSecure,
 kTRDC_MasterSecure1 }

TRDC secure attribute, the register bit TRDC_MDA_W0_0_DFMT0[SA], used for bus master domain assignment.
- enum `trdc_pid_domain_hit_config_t` {

kTRDC_pidDomainHitNone0,
 kTRDC_pidDomainHitNone1,
 kTRDC_pidDomainHitInclusive,
 kTRDC_pidDomainHitExclusive }

The configuration of domain hit evaluation of PID.
- enum `trdc_privilege_attr_t` {

kTRDC_ForceUser,
 kTRDC_ForcePrivilege,
 kTRDC_MasterPrivilege,
 kTRDC_MasterPrivilege1 }

TRDC privileged attribute, the register bit TRDC_MDA_W0_x_DFMT1[PA], used for non-processor bus master domain assignment.
- enum `trdc_pid_lock_t` {

kTRDC_PidUnlocked0,
 kTRDC_PidUnlocked1,
 kTRDC_PidUnlocked2,
 kTRDC_PidLocked }

PID lock configuration.
- enum `trdc_controller_t` {

```

kTRDC_MemBlockController0 = 0U,
kTRDC_MemBlockController1 = 1U,
kTRDC_MemBlockController2 = 2U,
kTRDC_MemBlockController3 = 3U,
kTRDC_MemRegionChecker0 = 4U,
kTRDC_MemRegionChecker1 = 5U,
kTRDC_MemRegionChecker2 = 6U,
kTRDC_MemRegionChecker3 = 7U,
kTRDC_MemRegionChecker4 = 8U,
kTRDC_MemRegionChecker5 = 9U,
kTRDC_MemRegionChecker6 = 10U }

```

TRDC controller definition for domain error check.

- enum `trdc_error_state_t` {
 kTRDC_ErrorStateNone = 0x00U,
 kTRDC_ErrorStateNone1 = 0x01U,
 kTRDC_ErrorStateSingle = 0x02U,
 kTRDC_ErrorStateMulti = 0x03U }

TRDC domain error state definition TRDC_MBCn_DERR_W1[EST] or TRDC_MRCn_DERR_W1[EST].

- enum `trdc_error_attr_t` {
 kTRDC_ErrorSecureUserInst = 0x00U,
 kTRDC_ErrorSecureUserData = 0x01U,
 kTRDC_ErrorSecurePrivilegeInst = 0x02U,
 kTRDC_ErrorSecurePrivilegeData = 0x03U,
 kTRDC_ErrorNonSecureUserInst = 0x04U,
 kTRDC_ErrorNonSecureUserData = 0x05U,
 kTRDC_ErrorNonSecurePrivilegeInst = 0x06U,
 kTRDC_ErrorNonSecurePrivilegeData = 0x07U }

TRDC domain error attribute definition TRDC_MBCn_DERR_W1[EATR] or TRDC_MRCn_DERR_W1[EATR].

- enum `trdc_error_type_t` {
 kTRDC_ErrorTypeRead = 0x00U,
 kTRDC_ErrorTypeWrite = 0x01U }

TRDC domain error access type definition TRDC_DERR_W1_n[ERW].

- enum `_trdc_region_descriptor` {

```
kTRDC_RegionDescriptor0 = (1U << 0U),
kTRDC_RegionDescriptor1 = (1U << 1U),
kTRDC_RegionDescriptor2 = (1U << 2U),
kTRDC_RegionDescriptor3 = (1U << 3U),
kTRDC_RegionDescriptor4 = (1U << 4U),
kTRDC_RegionDescriptor5 = (1U << 5U),
kTRDC_RegionDescriptor6 = (1U << 6U),
kTRDC_RegionDescriptor7 = (1U << 7U),
kTRDC_RegionDescriptor8 = (1U << 8U),
kTRDC_RegionDescriptor9 = (1U << 9U),
kTRDC_RegionDescriptor10 = (1U << 10U),
kTRDC_RegionDescriptor11 = (1U << 11U),
kTRDC_RegionDescriptor12 = (1U << 12U),
kTRDC_RegionDescriptor13 = (1U << 13U),
kTRDC_RegionDescriptor14 = (1U << 14U),
kTRDC_RegionDescriptor15 = (1U << 15U) }
```

The region descriptor enumeration, used to form a mask to set/clear the NSE bits for one or several regions.

- enum `_trdc_MRC_domain` {

```
kTRDC_MrcDomain0 = (1U << 0U),
kTRDC_MrcDomain1 = (1U << 1U),
kTRDC_MrcDomain2 = (1U << 2U),
kTRDC_MrcDomain3 = (1U << 3U),
kTRDC_MrcDomain4 = (1U << 4U),
kTRDC_MrcDomain5 = (1U << 5U),
kTRDC_MrcDomain6 = (1U << 6U),
kTRDC_MrcDomain7 = (1U << 7U),
kTRDC_MrcDomain8 = (1U << 8U),
kTRDC_MrcDomain9 = (1U << 9U),
kTRDC_MrcDomain10 = (1U << 10U),
kTRDC_MrcDomain11 = (1U << 11U),
kTRDC_MrcDomain12 = (1U << 12U),
kTRDC_MrcDomain13 = (1U << 13U),
kTRDC_MrcDomain14 = (1U << 14U),
kTRDC_MrcDomain15 = (1U << 15U) }
```

The MRC domain enumeration, used to form a mask to enable/disable the update or clear all NSE bits of one or several domains.

- enum `_trdc_MBC_domain` {

```
kTRDC_MbcDomain0 = (1U << 0U),
kTRDC_MbcDomain1 = (1U << 1U),
kTRDC_MbcDomain2 = (1U << 2U),
kTRDC_MbcDomain3 = (1U << 3U),
kTRDC_MbcDomain4 = (1U << 4U),
kTRDC_MbcDomain5 = (1U << 5U),
kTRDC_MbcDomain6 = (1U << 6U),
kTRDC_MbcDomain7 = (1U << 7U) }
```

The MBC domain enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several domains.

- enum _trdc_MBC_memory {

kTRDC_MbcSlaveMemory0 = (1U << 0U),

kTRDC_MbcSlaveMemory1 = (1U << 1U),

kTRDC_MbcSlaveMemory2 = (1U << 2U),

kTRDC_MbcSlaveMemory3 = (1U << 3U) }

The MBC slave memory enumeration, used to form a mask to enable/disable the update or clear NSE bits of one or several memory block.

- enum _trdc_MBC_bit {

kTRDC_MbcBit0 = (1U << 0U),

kTRDC_MbcBit1 = (1U << 1U),

kTRDC_MbcBit2 = (1U << 2U),

kTRDC_MbcBit3 = (1U << 3U),

kTRDC_MbcBit4 = (1U << 4U),

kTRDC_MbcBit5 = (1U << 5U),

kTRDC_MbcBit6 = (1U << 6U),

kTRDC_MbcBit7 = (1U << 7U),

kTRDC_MbcBit8 = (1U << 8U),

kTRDC_MbcBit9 = (1U << 9U),

kTRDC_MbcBit10 = (1U << 10U),

kTRDC_MbcBit11 = (1U << 11U),

kTRDC_MbcBit12 = (1U << 12U),

kTRDC_MbcBit13 = (1U << 13U),

kTRDC_MbcBit14 = (1U << 14U),

kTRDC_MbcBit15 = (1U << 15U),

kTRDC_MbcBit16 = (1U << 16U),

kTRDC_MbcBit17 = (1U << 17U),

kTRDC_MbcBit18 = (1U << 18U),

kTRDC_MbcBit19 = (1U << 19U),

kTRDC_MbcBit20 = (1U << 20U),

kTRDC_MbcBit21 = (1U << 21U),

kTRDC_MbcBit22 = (1U << 22U),

kTRDC_MbcBit23 = (1U << 23U),

kTRDC_MbcBit24 = (1U << 24U),

kTRDC_MbcBit25 = (1U << 25U),

kTRDC_MbcBit26 = (1U << 26U),

kTRDC_MbcBit27 = (1U << 27U),

kTRDC_MbcBit28 = (1U << 28U),

kTRDC_MbcBit29 = (1U << 29U),

kTRDC_MbcBit30 = (1U << 30U),

kTRDC_MbcBit31 = (1U << 31U) }

The MBC bit enumeration, used to form a mask to set/clear configured words' NSE.

Initialization and deinitialization

- void **TRDC_Init** (TRDC_Type *base)
Initializes the TRDC module.
- void **TRDC_Deinit** (TRDC_Type *base)
De-initializes the TRDC module.

Hardware configuration

- static uint8_t **TRDC_GetCurrentMasterDomainId** (TRDC_Type *base)
Gets the domain ID of the current bus master.
- void **TRDC_GetHardwareConfig** (TRDC_Type *base, trdc_hardware_config_t *config)
Gets the TRDC hardware configuration.

Master domain assignment

- static void **TRDC_SetDacGlobalValid** (TRDC_Type *base)
Sets the TRDC DAC(Domain Assignment Controllers) global valid.
- static void **TRDC_LockMasterDomainAssignment** (TRDC_Type *base, uint8_t master, uint8_t regNum)
Locks the bus master domain assignment register.
- static void **TRDC_SetMasterDomainAssignmentValid** (TRDC_Type *base, uint8_t master, uint8_t regNum, bool valid)
Sets the master domain assignment as valid or invalid.
- void **TRDC_GetDefaultProcessorDomainAssignment** (trdc_processor_domain_assignment_t *domainAssignment)
Gets the default master domain assignment for the processor bus master.
- void **TRDC_GetDefaultNonProcessorDomainAssignment** (trdc_non_processor_domain_assignment_t *domainAssignment)
Gets the default master domain assignment for non-processor bus master.
- void **TRDC_SetPid** (TRDC_Type *base, uint8_t master, const trdc_pid_config_t *pidConfig)
brief Sets the current Process identifier(PID) for processor core.
- static uint64_t **TRDC_GetActiveMasterPidMap** (TRDC_Type *base)
Gets the bit map of the bus master(s) that is(are) sourcing a PID register.
- void **TRDC_SetProcessorDomainAssignment** (TRDC_Type *base, uint8_t master, uint8_t regNum, const trdc_processor_domain_assignment_t *domainAssignment)
Sets the processor bus master domain assignment.
- static void **TRDC_EnableProcessorDomainAssignment** (TRDC_Type *base, uint8_t master, uint8_t regNum, bool enable)
Enables the processor bus master domain assignment.
- void **TRDC_SetNonProcessorDomainAssignment** (TRDC_Type *base, uint8_t master, const trdc_non_processor_domain_assignment_t *domainAssignment)
Sets the non-processor bus master domain assignment.

TZ-M configuration

- void **TRDC_GetDefaultIDAUConfig** (trdc_idau_config_t *idauConfiguration)
Gets the default IDAU(Implementation-Defined Attribution Unit) configuration.
- void **TRDC_SetIDAU** (TRDC_Type *base, const trdc_idau_config_t *idauConfiguration)
Sets the IDAU(Implementation-Defined Attribution Unit) control configuration.

FLW(Flash Logical Window) configuration

- static void `TRDC_EnableFlashLogicalWindow` (TRDC_Type *base, bool enable)
Enables/disables the FLW(flash logical window) function.
- static void `TRDC_LockFlashLogicalWindow` (TRDC_Type *base)
Locks FLW registers.
- static uint32_t `TRDC_GetFlashLogicalWindowPbase` (TRDC_Type *base)
Gets the FLW physical base address.
- static void `TRDC_GetSetFlashLogicalWindowSize` (TRDC_Type *base, uint16_t size)
Sets the FLW size.
- void `TRDC_GetDefaultFlashLogicalWindowConfig` (trdc_fwl_config_t *flwConfiguration)
Gets the default FLW(Flsh Logical Window) configuration.
- void `TRDC_SetFlashLogicalWindow` (TRDC_Type *base, const trdc_fwl_config_t *flw-Configuration)
Sets the FLW function's configuration.

Domain error check and clear

- status_t `TRDC_GetAndClearFirstDomainError` (TRDC_Type *base, trdc_domain_error_t *error)
Gets and clears the first domain error of the current domain.
- status_t `TRDC_GetAndClearFirstSpecificDomainError` (TRDC_Type *base, trdc_domain_error_t *error, uint8_t domainId)
Gets and clears the first domain error of the specific domain.

MRC configuration

- static void `TRDC_SetMrcGlobalValid` (TRDC_Type *base)
Sets the TRDC MRC(Memory Region Checkers) global valid.
- static uint8_t `TRDC_GetMrcRegionNumber` (TRDC_Type *base, uint8_t mrcIdx)
Gets the TRDC MRC(Memory Region Checkers) region number valid.
- void `TRDC_MrcSetMemoryAccessConfig` (TRDC_Type *base, const trdc_memory_access-control_config_t *config, uint8_t mrcIdx, uint8_t regIdx)
Sets the memory access configuration for one of the access control register of one MRC.
- void `TRDC_MrcEnableDomainNseUpdate` (TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask, bool enable)
Enables the update of the selected domains.
- void `TRDC_MrcRegionNseSet` (TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
Sets the NSE bits of the selected regions for domains.
- void `TRDC_MrcRegionNseClear` (TRDC_Type *base, uint8_t mrcIdx, uint16_t regionMask)
Clears the NSE bits of the selected regions for domains.
- void `TRDC_MrcDomainNseClear` (TRDC_Type *base, uint8_t mrcIdx, uint16_t domainMask)
Clears the NSE bits for all the regions of the selected domains.
- void `TRDC_MrcSetRegionDescriptorConfig` (TRDC_Type *base, const trdc_mrc_region-descriptor_config_t *config)
Sets the configuration for one of the region descriptor per domain per MRC instnce.

MBC configuration

- static void `TRDC_SetMbcGlobalValid` (TRDC_Type *base)
Sets the TRDC MBC(Memory Block Checkers) global valid.

- void [TRDC_GetMbcHardwareConfig](#) (TRDC_Type *base, [trdc_slave_memory_hardware_config_t](#) *config, uint8_t mbcIdx, uint8_t slvIdx)

Gets the hardware configuration of the one of two slave memories within each MBC(memory block checker).
- void [TRDC_MbcSetNseUpdateConfig](#) (TRDC_Type *base, const [trdc_mbc_nse_update_config_t](#) *config, uint8_t mbcIdx)

Sets the NSR update configuration for one of the MBC instance.
- void [TRDC_MbcWordNseSet](#) (TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Sets the NSE bits of the selected configuration words according to NSE update configuration.
- void [TRDC_MbcWordNseClear](#) (TRDC_Type *base, uint8_t mbcIdx, uint32_t bitMask)

Clears the NSE bits of the selected configuration words according to NSE update configuration.
- void [TRDC_MbcNseClearAll](#) (TRDC_Type *base, uint8_t mbcIdx, uint16_t domainMask, uint8_t slave)

Clears all configuration words' NSE bits of the selected domain and memory.
- void [TRDC_MbcSetMemoryAccessConfig](#) (TRDC_Type *base, const [trdc_memory_access_control_config_t](#) *config, uint8_t mbcIdx, uint8_t rgdIdx)

Sets the memory access configuration for one of the region descriptor of one MBC.
- void [TRDC_MbcSetMemoryBlockConfig](#) (TRDC_Type *base, const [trdc_mbc_memory_block_config_t](#) *config)

Sets the configuration for one of the memory block per domain per MBC instance.

41.3 Data Structure Documentation

41.3.1 struct trdc.hardware_config_t

Data Fields

- uint8_t **masterNumber**
Number of bus masters.
- uint8_t **domainNumber**
Number of domains.
- uint8_t **mbcNumber**
Number of MBCs.
- uint8_t **mrcNumber**
Number of MRCs.

Field Documentation

- (1) `uint8_t trdc_hardware_config_t::masterNumber`
- (2) `uint8_t trdc_hardware_config_t::domainNumber`
- (3) `uint8_t trdc_hardware_config_t::mbcNumber`
- (4) `uint8_t trdc_hardware_config_t::mrcNumber`

41.3.2 struct trdc_slave_memory_hardware_config_t**Data Fields**

- `uint32_t blockNum`
Number of blocks.
- `uint32_t blockSize`
Block size.

Field Documentation

- (1) `uint32_t trdc_slave_memory_hardware_config_t::blockNum`
- (2) `uint32_t trdc_slave_memory_hardware_config_t::blockSize`

41.3.3 struct trdc_processor_domain_assignment_t**Data Fields**

- `uint32_t domainId: 4U`
Domain ID.
- `uint32_t domainIdSelect: 2U`
Domain ID select method, see [trdc_did_sel_t](#).
- `uint32_t pidDomainHitConfig: 2U`
The configuration of the domain hit evaluation for PID, see [trdc_pid_domain_hit_config_t](#).
- `uint32_t pidMask: 6U`
The mask combined with PID, so multiple PID can be included as part of the domain hit determination.
- `uint32_t secureAttr: 2U`
Secure attribute, see [trdc_secure_attr_t](#).
- `uint32_t pid: 6U`
The process identifier, combined with pidMask to form the domain hit determination.
- `uint32_t __pad0__: 8U`
Reserved.
- `uint32_t lock: 1U`
Lock the register.
- `uint32_t __pad1__: 1U`
Reserved.

Field Documentation

- (1) `uint32_t trdc_processor_domain_assignment_t::domainId`
- (2) `uint32_t trdc_processor_domain_assignment_t::domainIdSelect`
- (3) `uint32_t trdc_processor_domain_assignment_t::pidDomainHitConfig`
- (4) `uint32_t trdc_processor_domain_assignment_t::pidMask`

Set to 0 to disable.

- (5) `uint32_t trdc_processor_domain_assignment_t::secureAttr`
- (6) `uint32_t trdc_processor_domain_assignment_t::pid`
- (7) `uint32_t trdc_processor_domain_assignment_t::__pad0__`
- (8) `uint32_t trdc_processor_domain_assignment_t::lock`
- (9) `uint32_t trdc_processor_domain_assignment_t::__pad1__`

41.3.4 struct trdc_non_processor_domain_assignment_t

Data Fields

- `uint32_t domainId`: 4U
Domain ID.
- `uint32_t privilegeAttr`: 2U
Privileged attribute, see [trdc_privilege_attr_t](#).
- `uint32_t secureAttr`: 2U
Secure attribute, see [trdc_secure_attr_t](#).
- `uint32_t bypassDomainId`: 1U
Bypass domain ID.
- `uint32_t __pad0__`: 21U
Reserved.
- `uint32_t lock`: 1U
Lock the register.
- `uint32_t __pad1__`: 1U
Reserved.

Field Documentation

- (1) `uint32_t trdc_non_processor_domain_assignment_t::domainId`
- (2) `uint32_t trdc_non_processor_domain_assignment_t::privilegeAttr`
- (3) `uint32_t trdc_non_processor_domain_assignment_t::secureAttr`
- (4) `uint32_t trdc_non_processor_domain_assignment_t::bypassDomainId`
- (5) `uint32_t trdc_non_processor_domain_assignment_t::__pad0__`
- (6) `uint32_t trdc_non_processor_domain_assignment_t::lock`
- (7) `uint32_t trdc_non_processor_domain_assignment_t::__pad1__`

41.3.5 struct trdc_pid_config_t**Data Fields**

- `uint32_t pid`: 6U
The process identifier of the executing task.
- `uint32_t __pad0__`: 23U
Reserved.
- `uint32_t lock`: 2U
How to lock the register, see [trdc_pid_lock_t](#).
- `uint32_t __pad1__`: 1U
Reserved.

Field Documentation

- (1) `uint32_t trdc_pid_config_t::pid`

The highest bit can be used to define secure/nonsecure attribute of the task.

- (2) `uint32_t trdc_pid_config_t::__pad0__`
- (3) `uint32_t trdc_pid_config_t::lock`
- (4) `uint32_t trdc_pid_config_t::__pad1__`

41.3.6 struct trdc_idau_config_t**Data Fields**

- `uint32_t __pad0__`: 8U
Reserved.
- `uint32_t lockSecureVTOR`: 1U
Disable writes to secure VTOR(Vector Table Offset Register).

- `uint32_t lockNonsecureVTOR`: 1U
Disable writes to non-secure VTOR, Application interrupt and Reset Control Registers.
- `uint32_t lockSecureMPU`: 1U
Disable writes to secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor in Secure state.
- `uint32_t lockNonsecureMPU`: 1U
Disable writes to non-secure MPU(Memory Protection Unit) from software or from a debug agent connected to the processor.
- `uint32_t lockSAU`: 1U
Disable writes to SAU(Security Attribution Unit) registers.
- `uint32_t __pad1__`: 19U
Reserved.

Field Documentation

- (1) `uint32_t trdc_idau_config_t::__pad0__`
- (2) `uint32_t trdc_idau_config_t::lockSecureVTOR`
- (3) `uint32_t trdc_idau_config_t::lockNonsecureVTOR`
- (4) `uint32_t trdc_idau_config_t::lockSecureMPU`
- (5) `uint32_t trdc_idau_config_t::lockNonsecureMPU`
- (6) `uint32_t trdc_idau_config_t::lockSAU`
- (7) `uint32_t trdc_idau_config_t::__pad1__`

41.3.7 struct trdc_flw_config_t

Data Fields

- `uint16_t blockCount`
Block count of the Flash Logic Window in 32KByte blocks.
- `uint32_t arrayBaseAddr`
Flash array base address of the Flash Logical Window.
- `bool lock`
Disable writes to FLW registers.
- `bool enable`
Enable FLW function.

Field Documentation

- (1) `uint16_t trdc_flw_config_t::blockCount`
- (2) `uint32_t trdc_flw_config_t::arrayBaseAddr`
- (3) `bool trdc_flw_config_t::lock`
- (4) `bool trdc_flw_config_t::enable`

41.3.8 struct trdc_domain_error_t**Data Fields**

- `trdc_controller_t controller`
Which controller captured access violation.
- `uint32_t address`
Access address that generated access violation.
- `trdc_error_state_t errorState`
Error state.
- `trdc_error_attr_t errorAttr`
Error attribute.
- `trdc_error_type_t errorType`
Error type.
- `uint8_t errorPort`
Error port.
- `uint8_t domainId`
Domain ID.
- `uint8_t slaveMemoryIdx`
The slave memory index.

Field Documentation

- (1) `trdc_controller_t trdc_domain_error_t::controller`
- (2) `uint32_t trdc_domain_error_t::address`
- (3) `trdc_error_state_t trdc_domain_error_t::errorState`
- (4) `trdc_error_attr_t trdc_domain_error_t::errorAttr`
- (5) `trdc_error_type_t trdc_domain_error_t::errorType`
- (6) `uint8_t trdc_domain_error_t::errorPort`
- (7) `uint8_t trdc_domain_error_t::domainId`
- (8) `uint8_t trdc_domain_error_t::slaveMemoryIdx`

Only apply when violation in MBC.

41.3.9 struct trdc_memory_access_control_config_t

Data Fields

- `uint32_t nonsecureUsrX`: 1U
Allow nonsecure user execute access.
- `uint32_t nonsecureUsrW`: 1U
Allow nonsecure user write access.
- `uint32_t nonsecureUsrR`: 1U
Allow nonsecure user read access.
- `uint32_t __pad0__`: 1U
Reserved.
- `uint32_t nonsecurePrivX`: 1U
Allow nonsecure privilege execute access.
- `uint32_t nonsecurePrivW`: 1U
Allow nonsecure privilege write access.
- `uint32_t nonsecurePrivR`: 1U
Allow nonsecure privilege read access.
- `uint32_t __pad1__`: 1U
Reserved.
- `uint32_t secureUsrX`: 1U
Allow secure user execute access.
- `uint32_t secureUsrW`: 1U
Allow secure user write access.
- `uint32_t secureUsrR`: 1U
Allow secure user read access.
- `uint32_t __pad2__`: 1U
Reserved.
- `uint32_t securePrivX`: 1U
Allow nonsecure privilege execute access.
- `uint32_t securePrivW`: 1U
Allow nonsecure privilege write access.
- `uint32_t securePrivR`: 1U
Allow nonsecure privilege read access.
- `uint32_t __pad3__`: 16U
Reserved.
- `uint32_t lock`: 1U
Lock the configuration until next reset, only apply to access control register 0.

Field Documentation

- (1) `uint32_t trdc_memory_access_control_config_t::nonsecureUsrX`
- (2) `uint32_t trdc_memory_access_control_config_t::nonsecureUsrW`
- (3) `uint32_t trdc_memory_access_control_config_t::nonsecureUsrR`
- (4) `uint32_t trdc_memory_access_control_config_t::__pad0__`
- (5) `uint32_t trdc_memory_access_control_config_t::nonsecurePrivX`
- (6) `uint32_t trdc_memory_access_control_config_t::nonsecurePrivW`
- (7) `uint32_t trdc_memory_access_control_config_t::nonsecurePrivR`
- (8) `uint32_t trdc_memory_access_control_config_t::__pad1__`
- (9) `uint32_t trdc_memory_access_control_config_t::secureUsrX`
- (10) `uint32_t trdc_memory_access_control_config_t::secureUsrW`
- (11) `uint32_t trdc_memory_access_control_config_t::secureUsrR`
- (12) `uint32_t trdc_memory_access_control_config_t::__pad2__`
- (13) `uint32_t trdc_memory_access_control_config_t::securePrivX`
- (14) `uint32_t trdc_memory_access_control_config_t::securePrivW`
- (15) `uint32_t trdc_memory_access_control_config_t::securePrivR`
- (16) `uint32_t trdc_memory_access_control_config_t::__pad3__`
- (17) `uint32_t trdc_memory_access_control_config_t::lock`

41.3.10 struct trdc_mrc_region_descriptor_config_t

Data Fields

- `uint8_t memoryAccessControlSelect`
Select one of the 8 access control policies for this region, for access control policies see [trdc_memory_access_control_config_t](#).
- `uint32_t startAddr`
Physical start address.
- `bool valid`
Lock the register.
- `bool nseEnable`
Enable non-secure accesses and disable secure accesses.
- `uint32_t endAddr`

- *Physical start address.*
- `uint8_t mrcIdx`
The index of the MRC for this configuration to take effect.
- `uint8_t domainIdx`
The index of the domain for this configuration to take effect.
- `uint8_t regionIdx`
The index of the region for this configuration to take effect.

Field Documentation

- (1) `uint8_t trdc_mrc_region_descriptor_config_t::memoryAccessControlSelect`
- (2) `uint32_t trdc_mrc_region_descriptor_config_t::startAddr`
- (3) `bool trdc_mrc_region_descriptor_config_t::valid`
- (4) `bool trdc_mrc_region_descriptor_config_t::nseEnable`
- (5) `uint32_t trdc_mrc_region_descriptor_config_t::endAddr`
- (6) `uint8_t trdc_mrc_region_descriptor_config_t::mrclidx`
- (7) `uint8_t trdc_mrc_region_descriptor_config_t::domainIdx`
- (8) `uint8_t trdc_mrc_region_descriptor_config_t::regionIdx`

41.3.11 struct trdc_mbc_nse_update_config_t

Data Fields

- `uint32_t autoIncrement`: 1U
Whether to increment the word index after current word is updated using this configuration.
- `uint32_t __pad0__`: 1U
Reserved.
- `uint32_t wordIdx`: 4U
MBC configuration word index to be updated.
- `uint32_t __pad1__`: 2U
Reserved.
- `uint32_t memorySelect`: 4U
Bit mask of the selected memory to be updated.
- `uint32_t __pad2__`: 4U
Reserved.
- `uint32_t domianSelect`: 16U
Bit mask of the selected domain to be updated.

Field Documentation

- (1) `uint32_t trdc_mbc_nse_update_config_t::autoIncrement`
- (2) `uint32_t trdc_mbc_nse_update_config_t::__pad0__`
- (3) `uint32_t trdc_mbc_nse_update_config_t::wordIdx`
- (4) `uint32_t trdc_mbc_nse_update_config_t::__pad1__`
- (5) `uint32_t trdc_mbc_nse_update_config_t::memorySelect`
`_trdc_MBC_memory.`
- (6) `uint32_t trdc_mbc_nse_update_config_t::__pad2__`
- (7) `uint32_t trdc_mbc_nse_update_config_t::domainSelect`
`_trdc_MBC_domain.`

41.3.12 struct trdc_mbc_memory_block_config_t**Data Fields**

- `uint32_t memoryAccessControlSelect: 3U`
Select one of the 8 access control policies for this memory block, for access control policies see [trdc_memory_access_control_config_t](#).
- `uint32_t nseEnable: 1U`
Enable non-secure accesses and disable secure accesses.
- `uint32_t mbcIdx: 4U`
The index of the MBC for this configuration to take effect.
- `uint32_t domainIdx: 8U`
The index of the domain for this configuration to take effect.
- `uint32_t slaveMemoryIdx: 8U`
The index of the slave memory for this configuration to take effect.
- `uint32_t memoryBlockIdx: 8U`
The index of the memory block for this configuration to take effect.

Field Documentation

- (1) `uint32_t trdc_mbc_memory_block_config_t::memoryAccessControlSelect`
- (2) `uint32_t trdc_mbc_memory_block_config_t::nseEnable`
- (3) `uint32_t trdc_mbc_memory_block_config_t::mbcIdx`
- (4) `uint32_t trdc_mbc_memory_block_config_t::domainIdx`
- (5) `uint32_t trdc_mbc_memory_block_config_t::slaveMemoryIdx`
- (6) `uint32_t trdc_mbc_memory_block_config_t::memoryBlockIdx`

41.4 Enumeration Type Documentation**41.4.1 enum trdc_did_sel_t**

Enumerator

kTRDC_DidMda Use MDAn[2:0] as DID.

kTRDC_DidInput Use the input DID (DID_in) as DID.

kTRDC_DidMdaAndInput Use MDAn[2] concatenated with DID_in[1:0] as DID.

kTRDC_DidReserved Reserved.

41.4.2 enum trdc_secure_attr_t

Enumerator

kTRDC_ForceSecure Force the bus attribute for this master to secure.

kTRDC_ForceNonSecure Force the bus attribute for this master to non-secure.

kTRDC_MasterSecure Use the bus master's secure/nonsecure attribute directly.

kTRDC_MasterSecure1 Use the bus master's secure/nonsecure attribute directly.

41.4.3 enum trdc_pid_domain_hit_config_t

Enumerator

kTRDC_pidDomainHitNone0 No PID is included in the domain hit evaluation.

kTRDC_pidDomainHitNone1 No PID is included in the domain hit evaluation.

kTRDC_pidDomainHitInclusive Domain hit when all the bits in PID is masked by the pidMask.

kTRDC_pidDomainHitExclusive Domain hit when not all the bits in PID is masked by the pidMask.

41.4.4 enum trdc_privilege_attr_t

Enumerator

kTRDC_ForceUser Force the bus attribute for this master to user.

kTRDC_ForcePrivilege Force the bus attribute for this master to privileged.

kTRDC_MasterPrivilege Use the bus master's attribute directly.

kTRDC_MasterPrivilege1 Use the bus master's attribute directly.

41.4.5 enum trdc_pid_lock_t

Enumerator

kTRDC_PidUnlocked0 The PID value can be updated by any secure priviledged write.

kTRDC_PidUnlocked1 The PID value can be updated by any secure priviledged write.

kTRDC_PidUnlocked2 The PID value can be updated by any secure priviledged write from the bus master that first configured this register.

kTRDC_PidLocked The PID value is locked until next reset.

41.4.6 enum trdc_controller_t

Each TRDC instance may have different MRC or MBC count, call TRDC_GetHardwareConfig to get the actual count.

Enumerator

kTRDC_MemBlockController0 Memory block checker 0.

kTRDC_MemBlockController1 Memory block checker 1.

kTRDC_MemBlockController2 Memory block checker 2.

kTRDC_MemBlockController3 Memory block checker 3.

kTRDC_MemRegionChecker0 Memory region checker 0.

kTRDC_MemRegionChecker1 Memory region checker 1.

kTRDC_MemRegionChecker2 Memory region checker 2.

kTRDC_MemRegionChecker3 Memory region checker 3.

kTRDC_MemRegionChecker4 Memory region checker 4.

kTRDC_MemRegionChecker5 Memory region checker 5.

kTRDC_MemRegionChecker6 Memory region checker 6.

41.4.7 enum trdc_error_state_t

Enumerator

kTRDC_ErrorStateNone No access violation detected.

kTRDC_ErrorStateNone1 No access violation detected.
kTRDC_ErrorStateSingle Single access violation detected.
kTRDC_ErrorStateMulti Multiple access violation detected.

41.4.8 enum trdc_error_attr_t

Enumerator

kTRDC_ErrorSecureUserInst Secure user mode, instruction fetch access.
kTRDC_ErrorSecureUserData Secure user mode, data access.
kTRDC_ErrorSecurePrivilegeInst Secure privileged mode, instruction fetch access.
kTRDC_ErrorSecurePrivilegeData Secure privileged mode, data access.
kTRDC_ErrorNonSecureUserInst NonSecure user mode, instruction fetch access.
kTRDC_ErrorNonSecureUserData NonSecure user mode, data access.
kTRDC_ErrorNonSecurePrivilegeInst NonSecure privileged mode, instruction fetch access.
kTRDC_ErrorNonSecurePrivilegeData NonSecure privileged mode, data access.

41.4.9 enum trdc_error_type_t

Enumerator

kTRDC_ErrorTypeRead Error occurs on read reference.
kTRDC_ErrorTypeWrite Error occurs on write reference.

41.4.10 enum _trdc_region_descriptor

Enumerator

kTRDC_RegionDescriptor0 Region descriptor 0.
kTRDC_RegionDescriptor1 Region descriptor 1.
kTRDC_RegionDescriptor2 Region descriptor 2.
kTRDC_RegionDescriptor3 Region descriptor 3.
kTRDC_RegionDescriptor4 Region descriptor 4.
kTRDC_RegionDescriptor5 Region descriptor 5.
kTRDC_RegionDescriptor6 Region descriptor 6.
kTRDC_RegionDescriptor7 Region descriptor 7.
kTRDC_RegionDescriptor8 Region descriptor 8.
kTRDC_RegionDescriptor9 Region descriptor 9.
kTRDC_RegionDescriptor10 Region descriptor 10.
kTRDC_RegionDescriptor11 Region descriptor 11.
kTRDC_RegionDescriptor12 Region descriptor 12.

kTRDC_RegionDescriptor13 Region descriptor 13.
kTRDC_RegionDescriptor14 Region descriptor 14.
kTRDC_RegionDescriptor15 Region descriptor 15.

41.4.11 enum _trdc_MRC_domain

Enumerator

kTRDC_MrcDomain0 Domain 0.
kTRDC_MrcDomain1 Domain 1.
kTRDC_MrcDomain2 Domain 2.
kTRDC_MrcDomain3 Domain 3.
kTRDC_MrcDomain4 Domain 4.
kTRDC_MrcDomain5 Domain 5.
kTRDC_MrcDomain6 Domain 6.
kTRDC_MrcDomain7 Domain 7.
kTRDC_MrcDomain8 Domain 8.
kTRDC_MrcDomain9 Domain 9.
kTRDC_MrcDomain10 Domain 10.
kTRDC_MrcDomain11 Domain 11.
kTRDC_MrcDomain12 Domain 12.
kTRDC_MrcDomain13 Domain 13.
kTRDC_MrcDomain14 Domain 14.
kTRDC_MrcDomain15 Domain 15.

41.4.12 enum _trdc_MBC_domain

Enumerator

kTRDC_MbcDomain0 Domain 0.
kTRDC_MbcDomain1 Domain 1.
kTRDC_MbcDomain2 Domain 2.
kTRDC_MbcDomain3 Domain 3.
kTRDC_MbcDomain4 Domain 4.
kTRDC_MbcDomain5 Domain 5.
kTRDC_MbcDomain6 Domain 6.
kTRDC_MbcDomain7 Domain 7.

41.4.13 enum _trdc_MBC_memory

Enumerator

kTRDC_MbcSlaveMemory0 Memory 0.

kTRDC_MbcSlaveMemory1 Memory 1.
kTRDC_MbcSlaveMemory2 Memory 2.
kTRDC_MbcSlaveMemory3 Memory 3.

41.4.14 enum _trdc_MBC_bit

Enumerator

kTRDC_MbcBit0 Bit 0.
kTRDC_MbcBit1 Bit 1.
kTRDC_MbcBit2 Bit 2.
kTRDC_MbcBit3 Bit 3.
kTRDC_MbcBit4 Bit 4.
kTRDC_MbcBit5 Bit 5.
kTRDC_MbcBit6 Bit 6.
kTRDC_MbcBit7 Bit 7.
kTRDC_MbcBit8 Bit 8.
kTRDC_MbcBit9 Bit 9.
kTRDC_MbcBit10 Bit 10.
kTRDC_MbcBit11 Bit 11.
kTRDC_MbcBit12 Bit 12.
kTRDC_MbcBit13 Bit 13.
kTRDC_MbcBit14 Bit 14.
kTRDC_MbcBit15 Bit 15.
kTRDC_MbcBit16 Bit 16.
kTRDC_MbcBit17 Bit 17.
kTRDC_MbcBit18 Bit 18.
kTRDC_MbcBit19 Bit 19.
kTRDC_MbcBit20 Bit 20.
kTRDC_MbcBit21 Bit 21.
kTRDC_MbcBit22 Bit 22.
kTRDC_MbcBit23 Bit 23.
kTRDC_MbcBit24 Bit 24.
kTRDC_MbcBit25 Bit 25.
kTRDC_MbcBit26 Bit 26.
kTRDC_MbcBit27 Bit 27.
kTRDC_MbcBit28 Bit 28.
kTRDC_MbcBit29 Bit 29.
kTRDC_MbcBit30 Bit 30.
kTRDC_MbcBit31 Bit 31.

41.5 Function Documentation

41.5.1 void TRDC_Init (TRDC_Type * *base*)

This function enables the TRDC clock.

Parameters

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

41.5.2 void TRDC_Deinit (TRDC_Type * *base*)

This function disables the TRDC clock.

Parameters

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

41.5.3 static uint8_t TRDC_GetCurrentMasterDomainId (TRDC_Type * *base*) [inline], [static]

Parameters

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

Returns

Domain ID of current bus master.

41.5.4 void TRDC_GetHardwareConfig (TRDC_Type * *base*, trdc_hardware_config_t * *config*)

This function gets the TRDC hardware configurations, including number of bus masters, number of domains, number of MRCs and number of PACs.

Parameters

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

41.5.5 static void TRDC_SetDacGlobalValid (TRDC_Type * *base*) [inline], [static]

Once enabled, it will remain enabled until next reset.

Parameters

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

41.5.6 static void TRDC_LockMasterDomainAssignment (TRDC_Type * *base*, uint8_t *master*, uint8_t *regNum*) [inline], [static]

This function locks the master domain assignment. After it is locked, the register can't be changed until next reset.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>master</i>	Which master to configure, refer to trdcx_master_t in processor header file, x is trdc instance.
<i>regNum</i>	Which register to configure, processor master can have more than one register for the MDAC configuration.
<i>assignIndex</i>	Which assignment register to lock.

41.5.7 static void TRDC_SetMasterDomainAssignmentValid (TRDC_Type * *base*, uint8_t *master*, uint8_t *regNum*, bool *valid*) [inline], [static]

This function sets the master domain assignment as valid or invalid.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>master</i>	Which master to configure.
<i>regNum</i>	Which register to configure, processor master can have more than one register for the MDAC configuration.
<i>assignIndex</i>	Index for the domain assignment register.
<i>valid</i>	True to set valid, false to set invalid.

41.5.8 void TRDC_GetDefaultProcessorDomainAssignment (trdc_processor_domain_assignment_t * *domainAssignment*)

This function gets the default master domain assignment for the processor bus master. It should only be used for the processor bus masters, such as CORE0. This function sets the assignment as follows:

```
* assignment->domainId      = 0U;
* assignment->domainIdSelect = kTRDC_DidMda;
* assignment->lock          = 0U;
*
```

Parameters

<i>domain-Assignment</i>	Pointer to the assignment structure.
--------------------------	--------------------------------------

41.5.9 void TRDC_GetDefaultNonProcessorDomainAssignment (trdc_non_processor_domain_assignment_t * *domainAssignment*)

This function gets the default master domain assignment for non-processor bus master. It should only be used for the non-processor bus masters, such as DMA. This function sets the assignment as follows:

```
* assignment->domainId      = 0U;
* assignment->privilegeAttr   = kTRDC_ForceUser;
* assignment->secureAttr     = kTRDC_ForceSecure;
* assignment->bypassDomainId = 0U;
* assignment->lock          = 0U;
*
```

Parameters

<i>domain-Assignment</i>	Pointer to the assignment structure.
--------------------------	--------------------------------------

41.5.10 void TRDC_SetPid (TRDC_Type * *base*, uint8_t *master*, const trdc_pid_config_t * *pidConfig*)

Each processor has a corresponding process identifier (PID) which can be used to group tasks into different domains. Secure privileged software saves and restores the PID as part of any context switch.

param base TRDC peripheral base address. param master Which processor master to configure, refer to trdc_master_t in processor header file. param pidConfig Pointer to the configuration structure.

41.5.11 static uint64_t TRDC_GetActiveMasterPidMap (TRDC_Type * *base*) [inline], [static]

This function sets the non-processor master domain assignment as valid.

Parameters

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

Returns

the bit map of the master(s). Bit 1 sets indicates bus master 1.

41.5.12 void TRDC_SetProcessorDomainAssignment (**TRDC_Type** * *base*, **uint8_t master**, **uint8_t regNum**, **const trdc_processor_domain_assignment_t * domainAssignment**)

This function sets the processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter `assignIndex` specifies which assignment register to set.

Example: Set domain assignment for core 0.

```
* trdc_processor_domain_assignment_t processorAssignment;
*
* TRDC_SetDefaultProcessorDomainAssignment (&processorAssignment);
*
* processorAssignment.domainId = 0;
* processorAssignment.xxx      = xxx;
* TRDC_SetMasterDomainAssignment (TRDC, &processorAssignment);
*
```

Parameters

<i>base</i>	TRDC peripheral base address.
<i>master</i>	Which master to configure, refer to <code>trdc_master_t</code> in processor header file.
<i>regNum</i>	Which register to configure, processor master can have more than one register for the MDAC configuration.
<i>domainAssignment</i>	Pointer to the assignment structure.

41.5.13 static void TRDC_EnableProcessorDomainAssignment (**TRDC_Type** * *base*, **uint8_t master**, **uint8_t regNum**, **bool enable**) [inline], [static]

Parameters

<i>base</i>	TRDC peripheral base address.
<i>master</i>	Which master to configure, refer to trdcx_master_t in processor header file, x is trdc instance.
<i>regNum</i>	Which register to configure, processor master can have more than one register.
<i>enable</i>	True to enable, false to disable.

41.5.14 void TRDC_SetNonProcessorDomainAssignment (TRDC_Type * *base*, uint8_t *master*, const trdc_non_processor_domain_assignment_t * *domainAssignment*)

This function sets the non-processor master domain assignment as valid. One bus master might have multiple domain assignment registers. The parameter *assignIndex* specifies which assignment register to set.

Example: Set domain assignment for DMA0.

```
* trdc_non_processor_domain_assignment_t nonProcessorAssignment;
*
* TRDC_GetDefaultNonProcessorDomainAssignment (&
*     nonProcessorAssignment);
* nonProcessorAssignment.domainId = 1;
* nonProcessorAssignment.xxxx      = xxxx;
*
* TRDC_SetMasterDomainAssignment (TRDC, kTrdcMasterDma0, 0U, &nonProcessorAssignment);
*
```

Parameters

<i>base</i>	TRDC peripheral base address.
<i>master</i>	Which master to configure, refer to trdc_master_t in processor header file.
<i>domainAssignment</i>	Pointer to the assignment structure.

41.5.15 void TRDC_GetDefaultIDAUConfig (trdc_idau_config_t * *idauConfiguration*)

```
* config->lockSecureVTOR      = false;
* config->lockNonsecureVTOR  = false;
* config->lockSecureMPU       = false;
* config->lockNonsecureMPU   = false;
* config->lockSAU            = false;
*
```

Parameters

<i>domain-Assignment</i>	Pointer to the configuration structure.
--------------------------	---

41.5.16 void TRDC_SetIDAU (TRDC_Type * *base*, const trdc_idau_config_t * *idauConfiguration*)

Example: Lock the secure and non-secure MPU registers.

```
* trdc_idau_config_t idauConfiguration;
*
* TRDC_GetDefaultIDAUConfig(&idauConfiguration);
*
* idauConfiguration.lockSecureMPU = true;
* idauConfiguration.lockNonsecureMPU = true;
* TRDC_SetIDAU(TRDC, &idauConfiguration);
*
```

Parameters

<i>base</i>	TRDC peripheral base address.
<i>domain-Assignment</i>	Pointer to the configuration structure.

41.5.17 static void TRDC_EnableFlashLogicalWindow (TRDC_Type * *base*, bool *enable*) [inline], [static]

Parameters

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

41.5.18 static void TRDC_LockFlashLogicalWindow (TRDC_Type * *base*) [inline], [static]

Once locked the registers can noy be updated until next reset.

Parameters

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

41.5.19 static uint32_t TRDC_GetFlashLogicalWindowPbase (TRDC_Type * *base*) [inline], [static]

Parameters

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

Returns

Physical address of the FLW function.

41.5.20 static void TRDC_GetSetFlashLogicalWindowSize (TRDC_Type * *base*, uint16_t *size*) [inline], [static]

Parameters

<i>base</i>	TRDC peripheral base address.
<i>size</i>	Size of the FLW in unit of 32k bytes.

41.5.21 void TRDC_GetDefaultFlashLogicalWindowConfig (trdc_flw_config_t * *flwConfiguration*)

```
* config->blockCount      = false;
* config->arrayBaseAddr = false;
* config->lock          = false;
* config->enable        = false;
*
```

Parameters

<i>flw- Configuration</i>	Pointer to the configuration structure.
-------------------------------	---

41.5.22 void TRDC_SetFlashLogicalWindow (**TRDC_Type** * *base*, const **trdc_flw_config_t** * *flwConfiguration*)

```
* trdc_flw_config_t flwConfiguration;
*
* TRDC_SetDefaultIDAUConfig(&flwConfiguration);
*
* flwConfiguration.blockCount = 32U;
* flwConfiguration.arrayBaseAddr = 0xFFFFFFFF;
* TRDC_SetIDAU(TRDC, &flwConfiguration);
*
```

Parameters

<i>base</i>	TRDC peripheral base address.
<i>flw-Configuration</i>	Pointer to the configuration structure.

41.5.23 status_t TRDC_GetAndClearFirstDomainError (**TRDC_Type** * *base*, **trdc_domain_error_t** * *error*)

This function gets the first access violation information for the current domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>error</i>	Pointer to the error information.

Returns

If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_NoData.

41.5.24 status_t TRDC_GetAndClearFirstSpecificDomainError (**TRDC_Type** * *base*, **trdc_domain_error_t** * *error*, **uint8_t** *domainId*)

This function gets the first access violation information for the specific domain and clears the pending flag. There might be multiple access violations pending for the current domain. This function only processes the first error.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>error</i>	Pointer to the error information.
<i>domainId</i>	The error of which domain to get and clear.

Returns

If the access violation is captured, this function returns the kStatus_Success. The error information can be obtained from the parameter error. If no access violation is captured, this function returns the kStatus_NoData.

41.5.25 static void TRDC_SetMrcGlobalValid (TRDC_Type * *base*) [inline], [static]

Once enabled, it will remain enabled until next reset.

Parameters

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

41.5.26 static uint8_t TRDC_GetMrcRegionNumber (TRDC_Type * *base*, uint8_t *mrcIdx*) [inline], [static]

Parameters

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

Returns

the region number of the given MRC instance

41.5.27 void TRDC_MrcSetMemoryAccessConfig (TRDC_Type * *base*, const trdc_memory_access_control_config_t * *config*, uint8_t *mrcIdx*, uint8_t *regIdx*)

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
* trdc_memory_access_control_config_t config;
*
* config.securePrivX = true;
* config.securePrivW = true;
* config.securePrivR = true;
* config.lock = true;
* TRDC_SetMrcMemoryAccess(TRDC, &config, 0, 1);
*
```

Parameters

<i>base</i>	TRDC peripheral base address.
<i>config</i>	Pointer to the configuration structure.
<i>mrcIdx</i>	MRC index.
<i>regIdx</i>	Register number.

41.5.28 void TRDC_MrcEnableDomainNseUpdate (TRDC_Type * *base*, uint8_t *mrcIdx*, uint16_t *domianMask*, bool *enable*)

After the domains' update are enabled, their regions' NSE bits can be set or clear.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>mrcIdx</i>	MRC index.
<i>domianMask</i>	Bit mask of the domains to be enabled.
<i>enable</i>	True to enable, false to disable.

41.5.29 void TRDC_MrcRegionNseSet (TRDC_Type * *base*, uint8_t *mrcIdx*, uint16_t *regionMask*)

This function sets the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>mrcIdx</i>	MRC index.

<i>regionMask</i>	Bit mask of the regions whose NSE bits to set.
-------------------	--

41.5.30 void TRDC_MrcRegionNseClear (**TRDC_Type** * *base*, **uint8_t** *mrcIdx*, **uint16_t** *regionMask*)

This function clears the NSE bits for the selected regions for the domains whose update are enabled.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>mrcIdx</i>	MRC index.
<i>regionMask</i>	Bit mask of the regions whose NSE bits to clear.

41.5.31 void TRDC_MrcDomainNseClear (**TRDC_Type** * *base*, **uint8_t** *mrcIdx*, **uint16_t** *domainMask*)

This function clears the NSE bits for all regions of selected domains whose update are enabled.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>mrcIdx</i>	MRC index.
<i>domainMask</i>	Bit mask of the domains whose NSE bits to clear.

41.5.32 void TRDC_MrcSetRegionDescriptorConfig (**TRDC_Type** * *base*, **const trdc_mrc_region_descriptor_config_t** * *config*)

This function sets the configuration for one of the region descriptor, including the start and end address of the region, memory access control policy and valid.

Parameters

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

<i>config</i>	Pointer to region descriptor configuration structure.
---------------	---

41.5.33 static void TRDC_SetMbcGlobalValid (TRDC_Type * *base*) [inline], [static]

Once enabled, it will remain enabled until next reset.

Parameters

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

41.5.34 void TRDC_GetMbcHardwareConfig (TRDC_Type * *base*, trdc_slave_memory_hardware_config_t * *config*, uint8_t *mbcIdx*, uint8_t *slvIdx*)

Parameters

<i>base</i>	TRDC peripheral base address.
<i>config</i>	Pointer to the structure to get the configuration.
<i>mbcIdx</i>	MBC number.
<i>slvIdx</i>	Slave number.

41.5.35 void TRDC_MbcSetNseUpdateConfig (TRDC_Type * *base*, const trdc_mbc_nse_update_config_t * *config*, uint8_t *mbcIdx*)

After set the NSE configuration, the configured memory area can be updateby NSE set/clear.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>config</i>	Pointer to NSE update configuration structure.
<i>mbcIdx</i>	MBC index.

41.5.36 **void TRDC_MbcWordNseSet (TRDC_Type * *base*, uint8_t *mbcIdx*, uint32_t *bitMask*)**

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>mbcIdx</i>	MBC index.
<i>bitMask</i>	Mask of the bits whose NSE bits to set.

41.5.37 void TRDC_MbcWordNseClear (TRDC_Type * *base*, uint8_t *mbcIdx*, uint32_t *bitMask*)

This function sets the NSE bits of the word for the configured regio, memory.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>mbcIdx</i>	MBC index.
<i>bitMask</i>	Mask of the bits whose NSE bits to clear.

41.5.38 void TRDC_MbcNseClearAll (TRDC_Type * *base*, uint8_t *mbcIdx*, uint16_t *domainMask*, uint8_t *slave*)

Parameters

<i>base</i>	TRDC peripheral base address.
<i>mbcIdx</i>	MBC index.
<i>domainMask</i>	Mask of the domains whose NSE bits to clear, 0b110 means clear domain 1&2.
<i>slaveMask</i>	Mask of the slaves whose NSE bits to clear, 0x11 means clear all slave 0&1's NSE bits.

41.5.39 void TRDC_MbcSetMemoryAccessConfig (TRDC_Type * *base*, const trdc_memory_access_control_config_t * *config*, uint8_t *mbcIdx*, uint8_t *rgdIdx*)

Example: Enable the secure operations and lock the configuration for MRC0 region 1.

```
* trdc_memory_access_control_config_t config;
*
* config.securePrivX = true;
* config.securePrivW = true;
* config.securePrivR = true;
```

```
* config.lock = true;
* TRDC_SetMbcMemoryAccess(TRDC, &config, 0, 1);
*
```

Parameters

<i>base</i>	TRDC peripheral base address.
<i>config</i>	Pointer to the configuration structure.
<i>mbcIdx</i>	MBC index.
<i>rgdIdx</i>	Region descriptor number.

41.5.40 void TRDC_MbcSetMemoryBlockConfig (TRDC_Type * *base*, const trdc_mbc_memory_block_config_t * *config*)

This function sets the configuration for one of the memory block, including the memory access control policy and nse enable.

Parameters

<i>base</i>	TRDC peripheral base address.
<i>config</i>	Pointer to memory block configuration structure.

Chapter 42

USDHC: Ultra Secured Digital Host Controller Driver

42.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Ultra Secured Digital Host Controller (USDHC) module of MCUXpresso SDK/i.MX devices.

42.2 Typical use case

42.2.1 USDHC Operation

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

Cache maintain capability

The uSDHC host controller is intergrated with ADMA to have better transfer performance, so to maintain data integrity during DMA operations on the platform that has cache, USDHC driver provide a cache maintain functionality by define: `FSL_SDK_ENABLE_DRIVER_CACHE_CONTROL = 1` It is suggest that the address of buffer used for read/write is align with cache line size.

Scatter gather transfer capability

The USDHC driver implement scatter gather transfer functionality, so application can submit uncontinuous data buffer in one transfer request by the scatter gather api, to have this feature, USDHC driver has below api `USDHC_TransferScatterGatherADMANonBlocking` This function suppprt scatter gather transfer and cover the functionality of `USDHC_TransferNonBlocking` also, but if application would like to use the function, please enable function macro firstly, since the scatter gather functionality is disabled by default.
`#define FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER 1`

Please note that once the macro is defined, the `USDHC_TransferNonBlocking` will be removed automatically.

Data Structures

- struct `usdhc_adma2_descriptor_t`
 Defines the ADMA2 descriptor structure. [More...](#)
- struct `usdhc_capability_t`
 USDHC capability information. [More...](#)
- struct `usdhc_boot_config_t`
 Data structure to configure the MMC boot feature. [More...](#)
- struct `usdhc_config_t`
 Data structure to initialize the USDHC. [More...](#)
- struct `usdhc_command_t`
 Card command descriptor. [More...](#)

- struct `usdhc_adma_config_t`
ADMA configuration. [More...](#)
- struct `usdhc_scatter_gather_data_list_t`
Card scatter gather data list. [More...](#)
- struct `usdhc_scatter_gather_data_t`
Card scatter gather data descriptor. [More...](#)
- struct `usdhc_scatter_gather_transfer_t`
usdhc scatter gather transfer. [More...](#)
- struct `usdhc_data_t`
Card data descriptor. [More...](#)
- struct `usdhc_transfer_t`
Transfer state. [More...](#)
- struct `usdhc_transfer_callback_t`
USDHC callback functions. [More...](#)
- struct `usdhc_handle_t`
USDHC handle. [More...](#)
- struct `usdhc_host_t`
USDHC host descriptor. [More...](#)

Macros

- #define `USDHC_MAX_BLOCK_COUNT` (`USDHC_BLK_ATT_BLKCNT_MASK >> USDHC_BLK_ATT_BLKCNT_SHIFT`)
Maximum block count can be set one time.
- #define `FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER_0U`
USDHC scatter gather feature control macro.
- #define `USDHC_ADMA1_ADDRESS_ALIGN` (4096U)
The alignment size for ADDRESS filed in ADMA1's descriptor.
- #define `USDHC_ADMA1_LENGTH_ALIGN` (4096U)
The alignment size for LENGTH field in ADMA1's descriptor.
- #define `USDHC_ADMA2_ADDRESS_ALIGN` (4U)
The alignment size for ADDRESS field in ADMA2's descriptor.
- #define `USDHC_ADMA2_LENGTH_ALIGN` (4U)
The alignment size for LENGTH filed in ADMA2's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT` (12U)
The bit shift for ADDRESS filed in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK` (0xFFFFFU)
The bit mask for ADDRESS field in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT` (12U)
The bit shift for LENGTH filed in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK` (0xFFFFU)
The mask for LENGTH field in ADMA1's descriptor.
- #define `USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (`USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK + 1U - 4096U`)
The maximum value of LENGTH filed in ADMA1's descriptor.
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT` (16U)
The bit shift for LENGTH field in ADMA2's descriptor.
- #define `USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK` (0xFFFFU)
The bit mask for LENGTH field in ADMA2's descriptor.
- #define `USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY` (`USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK - 3U`)

The maximum value of LENGTH field in ADMA2's descriptor.

Typedefs

- `typedef uint32_t usdhc_adma1_descriptor_t`
Defines the ADMA1 descriptor structure.
- `typedef status_t(* usdhc_transfer_function_t)(USDHC_Type *base, usdhc_transfer_t *content)`
USDHC transfer function.

Enumerations

- `enum {`
 - `kStatus_USDHC_BusyTransferring = MAKE_STATUS(kStatusGroup_USDHC, 0U),`
 - `kStatus_USDHC_PrepareAdmaDescriptorFailed = MAKE_STATUS(kStatusGroup_USDHC, 1U),`
 - `kStatus_USDHC_SendCommandFailed = MAKE_STATUS(kStatusGroup_USDHC, 2U),`
 - `kStatus_USDHC_TransferDataFailed = MAKE_STATUS(kStatusGroup_USDHC, 3U),`
 - `kStatus_USDHC_DMADataAddrNotAlign = MAKE_STATUS(kStatusGroup_USDHC, 4U),`
 - `kStatus_USDHC_ReTuningRequest = MAKE_STATUS(kStatusGroup_USDHC, 5U),`
 - `kStatus_USDHC_TuningError = MAKE_STATUS(kStatusGroup_USDHC, 6U),`
 - `kStatus_USDHC_NotSupport = MAKE_STATUS(kStatusGroup_USDHC, 7U),`
 - `kStatus_USDHC_TransferDataComplete = MAKE_STATUS(kStatusGroup_USDHC, 8U),`
 - `kStatus_USDHC_SendCommandSuccess = MAKE_STATUS(kStatusGroup_USDHC, 9U),`
 - `kStatus_USDHC_TransferDMAComplete = MAKE_STATUS(kStatusGroup_USDHC, 10U) }`

Enum _usdhc_status.
- `enum {`
 - `kUSDHC_SupportAdmaFlag = USDHC_HOST_CTRL_CAP_ADMAS_MASK,`
 - `kUSDHC_SupportHighSpeedFlag = USDHC_HOST_CTRL_CAP_HSS_MASK,`
 - `kUSDHC_SupportDmaFlag = USDHC_HOST_CTRL_CAP_DMAS_MASK,`
 - `kUSDHC_SupportSuspendResumeFlag = USDHC_HOST_CTRL_CAP_SRS_MASK,`
 - `kUSDHC_SupportV330Flag = USDHC_HOST_CTRL_CAP_VS33_MASK,`
 - `kUSDHC_SupportV300Flag = USDHC_HOST_CTRL_CAP_VS30_MASK,`
 - `kUSDHC_SupportV180Flag = USDHC_HOST_CTRL_CAP_VS18_MASK,`
 - `kUSDHC_Support4BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 0U),`
 - `kUSDHC_Support8BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 1U),`
 - `kUSDHC_SupportDDR50Flag = USDHC_HOST_CTRL_CAP_DDR50_SUPPORT_MASK,`
 - `kUSDHC_SupportSDR104Flag = USDHC_HOST_CTRL_CAP_SDR104_SUPPORT_MASK,`
 - `kUSDHC_SupportSDR50Flag = USDHC_HOST_CTRL_CAP_SDR50_SUPPORT_MASK }`

Enum _usdhc_capability_flag.
- `enum {`
 - `kUSDHC_WakeupEventOnCardInt = USDHC_PROT_CTRL_WECINT_MASK,`
 - `kUSDHC_WakeupEventOnCardInsert = USDHC_PROT_CTRL_WECINS_MASK,`
 - `kUSDHC_WakeupEventOnCardRemove = USDHC_PROT_CTRL_WECRM_MASK,`
 - `kUSDHC_WakeupEventsAll }`

Enum _usdhc_wakeup_event.
- `enum {`

```

kUSDHC_ResetAll = USDHC_SYS_CTRL_RSTA_MASK,
kUSDHC_ResetCommand = USDHC_SYS_CTRL_RSTC_MASK,
kUSDHC_ResetData = USDHC_SYS_CTRL_RSTD_MASK,
kUSDHC_ResetTuning = USDHC_SYS_CTRL_RSTT_MASK,
kUSDHC_ResetsAll = (kUSDHC_ResetAll | kUSDHC_ResetCommand | kUSDHC_ResetData |
kUSDHC_ResetTuning) }

Enum usdhc_reset.
• enum {
    kUSDHC_EnableDmaFlag = USDHC_MIX_CTRL_DMAEN_MASK,
    kUSDHC_CommandTypeSuspendFlag = USDHC_CMD_XFR_TYP_CMDTYP(1U),
    kUSDHC_CommandTypeResumeFlag = USDHC_CMD_XFR_TYP_CMDTYP(2U),
    kUSDHC_CommandTypeAbortFlag = USDHC_CMD_XFR_TYP_CMDTYP(3U),
    kUSDHC_EnableBlockCountFlag = USDHC_MIX_CTRL_BCEN_MASK,
    kUSDHC_EnableAutoCommand12Flag = USDHC_MIX_CTRL_AC12EN_MASK,
    kUSDHC_DataReadFlag = USDHC_MIX_CTRL_DTDSEL_MASK,
    kUSDHC_MultipleBlockFlag = USDHC_MIX_CTRL_MSBSEL_MASK,
    kUSDHC_EnableAutoCommand23Flag = USDHC_MIX_CTRL_AC23EN_MASK,
    kUSDHC_ResponseLength136Flag = USDHC_CMD_XFR_TYP_RSPTYP(1U),
    kUSDHC_ResponseLength48Flag = USDHC_CMD_XFR_TYP_RSPTYP(2U),
    kUSDHC_ResponseLength48BusyFlag = USDHC_CMD_XFR_TYP_RSPTYP(3U),
    kUSDHC_EnableCrcCheckFlag = USDHC_CMD_XFR_TYP_CCCEN_MASK,
    kUSDHC_EnableIndexCheckFlag = USDHC_CMD_XFR_TYP_CICEN_MASK,
    kUSDHC_DataPresentFlag = USDHC_CMD_XFR_TYP_DPSEL_MASK }

Enum usdhc_transfer_flag.
• enum {
    kUSDHC_CommandInhibitFlag = USDHC_PRES_STATE_CIHB_MASK,
    kUSDHC_DataInhibitFlag = USDHC_PRES_STATE_CDIHB_MASK,
    kUSDHC_DataLineActiveFlag = USDHC_PRES_STATE_DLAI_MASK,
    kUSDHC_SdClockStableFlag = USDHC_PRES_STATE_SDSTB_MASK,
    kUSDHC_WriteTransferActiveFlag = USDHC_PRES_STATE_WTA_MASK,
    kUSDHC_ReadTransferActiveFlag = USDHC_PRES_STATE_RTA_MASK,
    kUSDHC_BufferWriteEnableFlag = USDHC_PRES_STATE_BWEN_MASK,
    kUSDHC_BufferReadEnableFlag = USDHC_PRES_STATE_BREN_MASK,
    kUSDHC_ReTuningRequestFlag = USDHC_PRES_STATE_RTR_MASK,
    kUSDHC_DelaySettingFinishedFlag = USDHC_PRES_STATE_TSCD_MASK,
    kUSDHC_CardInsertedFlag = USDHC_PRES_STATE_CINST_MASK,
    kUSDHC_CommandLineLevelFlag = USDHC_PRES_STATE_CLSL_MASK,
    kUSDHC_Data0LineLevelFlag = 1U << USDHC_PRES_STATE_DLSL_SHIFT,
    kUSDHC_Data1LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 1U),
    kUSDHC_Data2LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 2U),
    kUSDHC_Data3LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 3U),
    kUSDHC_Data4LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 4U),
    kUSDHC_Data5LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 5U),
    kUSDHC_Data6LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 6U),
    kUSDHC_Data7LineLevelFlag = (int)(1U << (USDHC_PRES_STATE_DLSL_SHIFT + 7U)) }

```

- *Enum _usdhc_present_status_flag.*
- enum {

kUSDHC_CommandCompleteFlag = USDHC_INT_STATUS_CC_MASK,
 kUSDHC_DataCompleteFlag = USDHC_INT_STATUS_TC_MASK,
 kUSDHC_BlockGapEventFlag = USDHC_INT_STATUS_BGE_MASK,
 kUSDHC_DmaCompleteFlag = USDHC_INT_STATUS_DINT_MASK,
 kUSDHC_BufferWriteReadyFlag = USDHC_INT_STATUS_BWR_MASK,
 kUSDHC_BufferReadReadyFlag = USDHC_INT_STATUS_BRR_MASK,
 kUSDHC_CardInsertionFlag = USDHC_INT_STATUS_CINS_MASK,
 kUSDHC_CardRemovalFlag = USDHC_INT_STATUS_CRM_MASK,
 kUSDHC_CardInterruptFlag = USDHC_INT_STATUS_CINT_MASK,
 kUSDHC_ReTuningEventFlag = USDHC_INT_STATUS RTE_MASK,
 kUSDHC_TuningPassFlag = USDHC_INT_STATUS_TP_MASK,
 kUSDHC_TuningErrorFlag = USDHC_INT_STATUS_TNE_MASK,
 kUSDHC_CommandTimeoutFlag = USDHC_INT_STATUS_CTOE_MASK,
 kUSDHC_CommandCrcErrorFlag = USDHC_INT_STATUS_CCE_MASK,
 kUSDHC_CommandEndBitErrorFlag = USDHC_INT_STATUS_CEBE_MASK,
 kUSDHC_CommandIndexErrorFlag = USDHC_INT_STATUS_CIE_MASK,
 kUSDHC_DataTimeoutFlag = USDHC_INT_STATUS_DTOE_MASK,
 kUSDHC_DataCrcErrorFlag = USDHC_INT_STATUS_DCE_MASK,
 kUSDHC_DataEndBitErrorFlag = USDHC_INT_STATUS_DEBE_MASK,
 kUSDHC_AutoCommand12ErrorFlag = USDHC_INT_STATUS_AC12E_MASK,
 kUSDHC_DmaErrorFlag = USDHC_INT_STATUS_DMAE_MASK,
 kUSDHC_CommandErrorFlag,
 kUSDHC_DataErrorFlag,
 kUSDHC_ErrorFlag = (kUSDHC_CommandErrorFlag | kUSDHC_DataErrorFlag | kUSDHC_DmaErrorFlag),
 kUSDHC_DataFlag,
 kUSDHC_DataDMAFlag = (kUSDHC_DataCompleteFlag | kUSDHC_DataErrorFlag | kUSDHC_DmaErrorFlag),
 kUSDHC_CommandFlag = (kUSDHC_CommandErrorFlag | kUSDHC_CommandCompleteFlag),
 kUSDHC_CardDetectFlag = (kUSDHC_CardInsertionFlag | kUSDHC_CardRemovalFlag),
 kUSDHC_SDR104TuningFlag = (kUSDHC_TuningErrorFlag | kUSDHC_TuningPassFlag | kUSDHC_ReTuningEventFlag),
 kUSDHC_AllInterruptFlags }

- *Enum _usdhc_interrupt_status_flag.*
- enum {

kUSDHC_AutoCommand12NotExecutedFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12NE_MASK,
 kUSDHC_AutoCommand12TimeoutFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12TOE_MASK,
 kUSDHC_AutoCommand12EndBitErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12EBE_MASK,
 kUSDHC_AutoCommand12CrcErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12CE_

```

MASK,
kUSDHC_AutoCommand12IndexErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12IE-
_MASK,
kUSDHC_AutoCommand12NotIssuedFlag = USDHC_AUTOCMD12_ERR_STATUS_CNIBA-
C12E_MASK }

Enum_usdhc_auto_command12_error_status_flag.
• enum {
  kUSDHC_ExecuteTuning = USDHC_AUTOCMD12_ERR_STATUS_EXECUTE_TUNING_M-
ASK,
  kUSDHC_TuningSampleClockSel }

Enum_usdhc_standard_tuning.
• enum {
  kUSDHC_AdmaLengMismatchFlag = USDHC_ADMA_ERR_STATUSADMALME_MASK,
  kUSDHC_AdmaDescriptorErrorFlag = USDHC_ADMA_ERR_STATUSADMADCE_MASK }

Enum_usdhc_adma_error_status_flag.
• enum {
  kUSDHC_AdmaErrorStateStopDma = 0x00U,
  kUSDHC_AdmaErrorStateFetchDescriptor = 0x01U,
  kUSDHC_AdmaErrorStateChangeAddress = 0x02U,
  kUSDHC_AdmaErrorStateTransferData = 0x03U,
  kUSDHC_AdmaErrorStateInvalidLength = 0x04U,
  kUSDHC_AdmaErrorStateInvalidDescriptor = 0x08U,
  kUSDHC_AdmaErrorState }

Enum_usdhc_adma_error_state.
• enum {
  kUSDHC_ForceEventAutoCommand12NotExecuted,
  kUSDHC_ForceEventAutoCommand12Timeout = USDHC_FORCE_EVENT_FEVTAC12TOE-
MASK,
  kUSDHC_ForceEventAutoCommand12CrcError = USDHC_FORCE_EVENT_FEVTAC12CE-
MASK,
  kUSDHC_ForceEventEndBitError = USDHC_FORCE_EVENT_FEVTAC12EBE_MASK,
  kUSDHC_ForceEventAutoCommand12IndexError = USDHC_FORCE_EVENT_FEVTAC12IE-
MASK,
  kUSDHC_ForceEventAutoCommand12NotIssued = USDHC_FORCE_EVENT_FEVTCNIBA-
C12E_MASK,
  kUSDHC_ForceEventCommandTimeout = USDHC_FORCE_EVENT_FEVTCTOE_MASK,
  kUSDHC_ForceEventCommandCrcError = USDHC_FORCE_EVENT_FEVTCCE_MASK,
  kUSDHC_ForceEventCommandEndBitError = USDHC_FORCE_EVENT_FEVTCCEBE_MASK,
  kUSDHC_ForceEventCommandIndexError = USDHC_FORCE_EVENT_FEVTCIE_MASK,
  kUSDHC_ForceEventDataTimeout = USDHC_FORCE_EVENT_FEVTDTOE_MASK,
  kUSDHC_ForceEventDataCrcError = USDHC_FORCE_EVENT_FEVTDCE_MASK,
  kUSDHC_ForceEventDataEndBitError = USDHC_FORCE_EVENT_FEVTDEBE_MASK,
  kUSDHC_ForceEventAutoCommand12Error = USDHC_FORCE_EVENT_FEVTAC12E_MAS-
}

```

- K,
- `kUSDHC_ForceEventCardInt` = (int)USDHC_FORCE_EVENT_FEVTCINT_MASK,
 - `kUSDHC_ForceEventDmaError` = USDHC_FORCE_EVENT_FEVTDMAE_MASK,
 - `kUSDHC_ForceEventTuningError` = USDHC_FORCE_EVENT_FEVTTNE_MASK,
 - `kUSDHC_ForceEventsAll` }
- Enum usdhc_force_event.*
- enum `usdhc_transfer_direction_t` {
 - `kUSDHC_TransferDirectionReceive` = 1U,
 - `kUSDHC_TransferDirectionSend` = 0U }
- Data transfer direction.*
- enum `usdhc_data_bus_width_t` {
 - `kUSDHC_DataBusWidth1Bit` = 0U,
 - `kUSDHC_DataBusWidth4Bit` = 1U,
 - `kUSDHC_DataBusWidth8Bit` = 2U }
- Data transfer width.*
- enum `usdhc_endian_mode_t` {
 - `kUSDHC_EndianModeBig` = 0U,
 - `kUSDHC_EndianModeHalfWordBig` = 1U,
 - `kUSDHC_EndianModeLittle` = 2U }
- Endian mode.*
- enum `usdhc_dma_mode_t` {
 - `kUSDHC_DmaModeSimple` = 0U,
 - `kUSDHC_DmaModeAdma1` = 1U,
 - `kUSDHC_DmaModeAdma2` = 2U,
 - `kUSDHC_ExternalDMA` = 3U }
- DMA mode.*
- enum {
 - `kUSDHC_StopAtBlockGapFlag` = USDHC_PROT_CTRL_SABGREQ_MASK,
 - `kUSDHC_ReadWaitControlFlag` = USDHC_PROT_CTRL_RWCTL_MASK,
 - `kUSDHC InterruptAtBlockGapFlag` = USDHC_PROT_CTRL_IABG_MASK,
 - `kUSDHC_ReadDoneNo8CLK` = USDHC_PROT_CTRL_RD_DONE_NO_8CLK_MASK,
 - `kUSDHC_ExactBlockNumberReadFlag` = USDHC_PROT_CTRL_NON_EXACT_BLK_RD_M-ASK }
- Enum usdhc_sdio_control_flag.*
- enum `usdhc_boot_mode_t` {
 - `kUSDHC_BootModeNormal` = 0U,
 - `kUSDHC_BootModeAlternative` = 1U }
- MMC card boot mode.*
- enum `usdhc_card_command_type_t` {
 - `kCARD_CommandTypeNormal` = 0U,
 - `kCARD_CommandTypeSuspend` = 1U,
 - `kCARD_CommandTypeResume` = 2U,
 - `kCARD_CommandTypeAbort` = 3U,
 - `kCARD_CommandTypeEmpty` = 4U }
- The command type.*
- enum `usdhc_card_response_type_t` {

```
kCARD_ResponseNone = 0U,
kCARD_ResponseTypeR1 = 1U,
kCARD_ResponseTypeR1b = 2U,
kCARD_ResponseTypeR2 = 3U,
kCARD_ResponseTypeR3 = 4U,
kCARD_ResponseTypeR4 = 5U,
kCARD_ResponseTypeR5 = 6U,
kCARD_ResponseTypeR5b = 7U,
kCARD_ResponseTypeR6 = 8U,
kCARD_ResponseTypeR7 = 9U }
```

The command response type.

- enum {


```
kUSDHC_Adma1DescriptorValidFlag = (1U << 0U),
kUSDHC_Adma1DescriptorEndFlag = (1U << 1U),
kUSDHC_Adma1DescriptorInterruptFlag = (1U << 2U),
kUSDHC_Adma1DescriptorActivity1Flag = (1U << 4U),
kUSDHC_Adma1DescriptorActivity2Flag = (1U << 5U),
kUSDHC_Adma1DescriptorTypeNop = (kUSDHC_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeTransfer = (kUSDHC_Adma1DescriptorActivity2Flag | kUSDHC_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeLink,
kUSDHC_Adma1DescriptorTypeSetLength = (kUSDHC_Adma1DescriptorActivity1Flag | kUSDHC_Adma1DescriptorValidFlag) }
```

Enum_usdhc_adma1_descriptor_flag.
- enum {


```
kUSDHC_Adma2DescriptorValidFlag = (1U << 0U),
kUSDHC_Adma2DescriptorEndFlag = (1U << 1U),
kUSDHC_Adma2DescriptorInterruptFlag = (1U << 2U),
kUSDHC_Adma2DescriptorActivity1Flag = (1U << 4U),
kUSDHC_Adma2DescriptorActivity2Flag = (1U << 5U),
kUSDHC_Adma2DescriptorTypeNop = (kUSDHC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeReserved = (kUSDHC_Adma2DescriptorActivity1Flag | kUSDHC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeTransfer = (kUSDHC_Adma2DescriptorActivity2Flag | kUSDHC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeLink }
```

Enum_usdhc_adma2_descriptor_flag.
- enum {


```
kUSDHC_AdmaDescriptorSingleFlag = 0U,
kUSDHC_AdmaDescriptorMultipleFlag }
```

Enum_usdhc_adma_flag.
- enum usdhc_burst_len_t {


```
kUSDHC_EnBurstLenForINCR = 0x01U,
kUSDHC_EnBurstLenForINCR4816 = 0x02U,
kUSDHC_EnBurstLenForINCR4816WRAP = 0x04U }
```

DMA transfer burst len config.

- enum {

 kUSDHC_TransferDataNormal = 0U,
 kUSDHC_TransferDataTuning = 1U,
 kUSDHC_TransferDataBoot = 2U,
 kUSDHC_TransferDataBootcontinuous = 3U }
- Enum usdhc_transfer_data_type.*

Driver version

- #define **FSL_USDHC_DRIVER_VERSION** (**MAKE_VERSION**(2U, 8U, 4U))
Driver version 2.8.4.

Initialization and deinitialization

- void **USDHC_Init** (USDHC_Type *base, const **usdhc_config_t** *config)
USDHC module initialization function.
- void **USDHC_Deinit** (USDHC_Type *base)
Deinitializes the USDHC.
- bool **USDHC_Reset** (USDHC_Type *base, uint32_t mask, uint32_t timeout)
Resets the USDHC.

DMA Control

- status_t **USDHC_SetAdmaTableConfig** (USDHC_Type *base, **usdhc_adma_config_t** *dmaConfig, **usdhc_data_t** *dataConfig, uint32_t flags)
Sets the DMA descriptor table configuration.
- status_t **USDHC_SetInternalDmaConfig** (USDHC_Type *base, **usdhc_adma_config_t** *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)
Internal DMA configuration.
- status_t **USDHC_SetADMA2Descriptor** (uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)
Sets the ADMA2 descriptor table configuration.
- status_t **USDHC_SetADMA1Descriptor** (uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)
Sets the ADMA1 descriptor table configuration.
- static void **USDHC_EnableInternalDMA** (USDHC_Type *base, bool enable)
Enables internal DMA.

Interrupts

- static void **USDHC_EnableInterruptStatus** (USDHC_Type *base, uint32_t mask)
Enables the interrupt status.
- static void **USDHC_DisableInterruptStatus** (USDHC_Type *base, uint32_t mask)
Disables the interrupt status.
- static void **USDHC_EnableInterruptSignal** (USDHC_Type *base, uint32_t mask)
Enables the interrupt signal corresponding to the interrupt status flag.
- static void **USDHC_DisableInterruptSignal** (USDHC_Type *base, uint32_t mask)
Disables the interrupt signal corresponding to the interrupt status flag.

Status

- static uint32_t **USDHC_GetEnabledInterruptStatusFlags** (USDHC_Type *base)
Gets the enabled interrupt status.
- static uint32_t **USDHC_GetInterruptStatusFlags** (USDHC_Type *base)
Gets the current interrupt status.
- static void **USDHC_ClearInterruptStatusFlags** (USDHC_Type *base, uint32_t mask)
Clears a specified interrupt status.
- static uint32_t **USDHC_GetAutoCommand12ErrorStatusFlags** (USDHC_Type *base)
Gets the status of auto command 12 error.
- static uint32_t **USDHC_GetAdmaErrorStatusFlags** (USDHC_Type *base)
Gets the status of the ADMA error.
- static uint32_t **USDHC_GetPresentStatusFlags** (USDHC_Type *base)
Gets a present status.

Bus Operations

- void **USDHC_GetCapability** (USDHC_Type *base, usdhc_capability_t *capability)
Gets the capability information.
- static void **USDHC_ForceClockOn** (USDHC_Type *base, bool enable)
Forces the card clock on.
- uint32_t **USDHC_SetSdClock** (USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)
Sets the SD bus clock frequency.
- bool **USDHC_SetCardActive** (USDHC_Type *base, uint32_t timeout)
Sends 80 clocks to the card to set it to the active state.
- static void **USDHC_AssertHardwareReset** (USDHC_Type *base, bool high)
Triggers a hardware reset.
- static void **USDHC_SetDataBusWidth** (USDHC_Type *base, usdhc_data_bus_width_t width)
Sets the data transfer width.
- static void **USDHC_WriteData** (USDHC_Type *base, uint32_t data)
Fills the data port.
- static uint32_t **USDHC_ReadData** (USDHC_Type *base)
Retrieves the data from the data port.
- void **USDHC_SendCommand** (USDHC_Type *base, usdhc_command_t *command)
Sends command function.
- static void **USDHC_EnableWakeUpEvent** (USDHC_Type *base, uint32_t mask, bool enable)
Enables or disables a wakeup event in low-power mode.
- static void **USDHC_CardDetectByData3** (USDHC_Type *base, bool enable)
Detects card insert status.
- static bool **USDHC_DetectCardInsert** (USDHC_Type *base)
Detects card insert status.
- static void **USDHC_EnableSdioControl** (USDHC_Type *base, uint32_t mask, bool enable)
Enables or disables the SDIO card control.
- static void **USDHC_SetContinueRequest** (USDHC_Type *base)
Restarts a transaction which has stopped at the block GAP for the SDIO card.
- static void **USDHC_RequestStopAtBlockGap** (USDHC_Type *base, bool enable)
Request stop at block gap function.
- void **USDHC_SetMmcBootConfig** (USDHC_Type *base, const usdhc_boot_config_t *config)
Configures the MMC boot feature.
- static void **USDHC_EnableMmcBoot** (USDHC_Type *base, bool enable)
Enables or disables the mmc boot mode.

- static void **USDHC_SetForceEvent** (USDHC_Type *base, uint32_t mask)

Forces generating events according to the given mask.
- static void **USDHC_SelectVoltage** (USDHC_Type *base, bool en18v)

Selects the USDHC output voltage.
- static bool **USDHC_RequestTuningForSDR50** (USDHC_Type *base)

Checks the SDR50 mode request tuning bit.
- static bool **USDHC_RequestReTuning** (USDHC_Type *base)

Checks the request re-tuning bit.
- static void **USDHC_EnableAutoTuning** (USDHC_Type *base, bool enable)

The SDR104 mode auto tuning enable and disable.
- void **USDHC_EnableAutoTuningForCmdAndData** (USDHC_Type *base)

The auto tuning enable for CMD/DATA line.
- void **USDHC_EnableManualTuning** (USDHC_Type *base, bool enable)

Manual tuning trigger or abort.
- static uint32_t **USDHC_GetTuningDelayStatus** (USDHC_Type *base)

Get the tuning delay cell setting.
- status_t **USDHC_SetTuningDelay** (USDHC_Type *base, uint32_t preDelay, uint32_t outDelay, uint32_t postDelay)

The tuning delay cell setting.
- status_t **USDHC_AdjustDelayForManualTuning** (USDHC_Type *base, uint32_t delay)

Adjusts delay for manual tuning.
- static void **USDHC_SetStandardTuningCounter** (USDHC_Type *base, uint8_t counter)

set tuning counter tuning.
- void **USDHC_EnableStandardTuning** (USDHC_Type *base, uint32_t tuningStartTap, uint32_t step, bool enable)

The enable standard tuning function.
- static uint32_t **USDHC_GetExecuteStdTuningStatus** (USDHC_Type *base)

Gets execute STD tuning status.
- static uint32_t **USDHC_CheckStdTuningResult** (USDHC_Type *base)

Checks STD tuning result.
- static uint32_t **USDHC_CheckTuningError** (USDHC_Type *base)

Checks tuning error.
- void **USDHC_EnableDDRMode** (USDHC_Type *base, bool enable, uint32_t nibblePos)

The enable/disable DDR mode.
- static void **USDHC_EnableHS400Mode** (USDHC_Type *base, bool enable)

The enable/disable HS400 mode.
- static void **USDHC_ResetStrobeDLL** (USDHC_Type *base)

Resets the strobe DLL.
- static void **USDHC_EnableStrobeDLL** (USDHC_Type *base, bool enable)

Enables/disables the strobe DLL.
- void **USDHC_ConfigStrobeDLL** (USDHC_Type *base, uint32_t delayTarget, uint32_t updateInterval)

Configs the strobe DLL delay target and update interval.
- static void **USDHC_SetStrobeDlOverride** (USDHC_Type *base, uint32_t delayTaps)

*Enables manual override for slave delay chain using **STROBE_SLV_OVERRIDE_VAL**.*
- static uint32_t **USDHC_GetStrobeDLLStatus** (USDHC_Type *base)

Gets the strobe DLL status.
- void **USDHC_SetDataConfig** (USDHC_Type *base, usdhc_transfer_direction_t dataDirection, uint32_t blockCount, uint32_t blockSize)

USDHC data configuration.

Transactional functions

- void [USDHC_TransferCreateHandle](#) (USDHC_Type *base, usdhc_handle_t *handle, const [usdhc_transfer_callback_t](#) *callback, void *userData)
Creates the USDHC handle.
- status_t [USDHC_TransferNonBlocking](#) (USDHC_Type *base, usdhc_handle_t *handle, [usdhc_adma_config_t](#) *dmaConfig, [usdhc_transfer_t](#) *transfer)
Transfers the command/data using an interrupt and an asynchronous method.
- status_t [USDHC_TransferBlocking](#) (USDHC_Type *base, [usdhc_adma_config_t](#) *dmaConfig, [usdhc_transfer_t](#) *transfer)
Transfers the command/data using a blocking method.
- void [USDHC_TransferHandleIRQ](#) (USDHC_Type *base, usdhc_handle_t *handle)
IRQ handler for the USDHC.

42.3 Data Structure Documentation

42.3.1 struct usdhc_adma2_descriptor_t

Data Fields

- uint32_t [attribute](#)
The control and status field.
- const uint32_t * [address](#)
The address field.

Field Documentation

- (1) uint32_t [usdhc_adma2_descriptor_t::attribute](#)
- (2) const uint32_t* [usdhc_adma2_descriptor_t::address](#)

42.3.2 struct usdhc_capability_t

Defines a structure to save the capability information of USDHC.

Data Fields

- uint32_t [sdVersion](#)
Support SD card/sdio version.
- uint32_t [mmcVersion](#)
Support EMMC card version.
- uint32_t [maxBlockLength](#)
Maximum block length united as byte.
- uint32_t [maxBlockCount](#)
Maximum block count can be set one time.
- uint32_t [flags](#)
Capability flags to indicate the support information([_usdhc_capability_flag](#)).

Field Documentation

- (1) `uint32_t usdhc_capability_t::sdVersion`
- (2) `uint32_t usdhc_capability_t::mmcVersion`
- (3) `uint32_t usdhc_capability_t::maxBlockLength`
- (4) `uint32_t usdhc_capability_t::maxBlockCount`
- (5) `uint32_t usdhc_capability_t::flags`

42.3.3 struct usdhc_boot_config_t**Data Fields**

- `uint32_t ackTimeoutCount`
Timeout value for the boot ACK.
- `usdhc_boot_mode_t bootMode`
Boot mode selection.
- `uint32_t blockCount`
Stop at block gap value of automatic mode.
- `size_t blockSize`
Block size.
- `bool enableBootAck`
Enable or disable boot ACK.
- `bool enableAutoStopAtBlockGap`
Enable or disable auto stop at block gap function in boot period.

Field Documentation

- (1) `uint32_t usdhc_boot_config_t::ackTimeoutCount`

The available range is 0 ~ 15.

- (2) `usdhc_boot_mode_t usdhc_boot_config_t::bootMode`

- (3) `uint32_t usdhc_boot_config_t::blockCount`

Available range is 0 ~ 65535.

- (4) `size_t usdhc_boot_config_t::blockSize`
- (5) `bool usdhc_boot_config_t::enableBootAck`
- (6) `bool usdhc_boot_config_t::enableAutoStopAtBlockGap`

42.3.4 struct usdhc_config_t

Data Fields

- `uint32_t dataTimeout`
Data timeout value.
- `usdhc_endian_mode_t endianMode`
Endian mode.
- `uint8_t readWatermarkLevel`
Watermark level for DMA read operation.
- `uint8_t writeWatermarkLevel`
Watermark level for DMA write operation.

Field Documentation

- (1) `uint32_t usdhc_config_t::dataTimeout`
- (2) `usdhc_endian_mode_t usdhc_config_t::endianMode`
- (3) `uint8_t usdhc_config_t::readWatermarkLevel`

Available range is 1 ~ 128.

- (4) `uint8_t usdhc_config_t::writeWatermarkLevel`

Available range is 1 ~ 128.

42.3.5 struct usdhc_command_t

Defines card command-related attribute.

Data Fields

- `uint32_t index`
Command index.
- `uint32_t argument`
Command argument.
- `usdhc_card_command_type_t type`
Command type.
- `usdhc_card_response_type_t responseType`
Command response type.

- `uint32_t response [4U]`
Response for this command.
- `uint32_t responseErrorFlags`
Response error flag, which need to check the command response.
- `uint32_t flags`
Cmd flags.

Field Documentation

- (1) `uint32_t usdhc_command_t::index`
- (2) `uint32_t usdhc_command_t::argument`
- (3) `usdhc_card_command_type_t usdhc_command_t::type`
- (4) `usdhc_card_response_type_t usdhc_command_t::responseType`
- (5) `uint32_t usdhc_command_t::response[4U]`
- (6) `uint32_t usdhc_command_t::responseErrorFlags`
- (7) `uint32_t usdhc_command_t::flags`

42.3.6 struct usdhc_adma_config_t**Data Fields**

- `usdhc_dma_mode_t dmaMode`
DMA mode.
- `uint32_t * admaTable`
ADMA table address, can't be null if transfer way is ADMA1/ADMA2.
- `uint32_t admaTableWords`
ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

Field Documentation

- (1) `usdhc_dma_mode_t usdhc_adma_config_t::dmaMode`
- (2) `uint32_t* usdhc_adma_config_t::admaTable`
- (3) `uint32_t usdhc_adma_config_t::admaTableWords`

42.3.7 struct usdhc_scatter_gather_data_list_t

Allow application register uncontinuous data buffer for data transfer.

42.3.8 struct usdhc_scatter_gather_data_t

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Data Fields

- bool `enableAutoCommand12`
Enable auto CMD12.
- bool `enableAutoCommand23`
Enable auto CMD23.
- bool `enableIgnoreError`
Enable to ignore error event to read/write all the data.
- `usdhc_transfer_direction_t dataDirection`
data direction
- `uint8_t dataType`
this is used to distinguish the normal/tuning/boot data.
- `size_t blockSize`
Block size.
- `usdhc_scatter_gather_data_list_t sgData`
scatter gather data

Field Documentation

- (1) `bool usdhc_scatter_gather_data_t::enableAutoCommand12`
- (2) `bool usdhc_scatter_gather_data_t::enableAutoCommand23`
- (3) `bool usdhc_scatter_gather_data_t::enableIgnoreError`
- (4) `uint8_t usdhc_scatter_gather_data_t::dataType`
- (5) `size_t usdhc_scatter_gather_data_t::blockSize`

42.3.9 struct usdhc_scatter_gather_transfer_t

Data Fields

- `usdhc_scatter_gather_data_t * data`
Data to transfer.
- `usdhc_command_t * command`
Command to send.

Field Documentation

- (1) `usdhc_scatter_gather_data_t*` `usdhc_scatter_gather_transfer_t::data`
- (2) `usdhc_command_t*` `usdhc_scatter_gather_transfer_t::command`

42.3.10 struct usdhc_data_t

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Data Fields

- bool `enableAutoCommand12`
Enable auto CMD12.
- bool `enableAutoCommand23`
Enable auto CMD23.
- bool `enableIgnoreError`
Enable to ignore error event to read/write all the data.
- `uint8_t dataType`
this is used to distinguish the normal/tuning/boot data.
- `size_t blockSize`
Block size.
- `uint32_t blockCount`
Block count.
- `uint32_t * rxData`
Buffer to save data read.
- `const uint32_t * txData`
Data buffer to write.

Field Documentation

- (1) **bool usdhc_data_t::enableAutoCommand12**
- (2) **bool usdhc_data_t::enableAutoCommand23**
- (3) **bool usdhc_data_t::enableIgnoreError**
- (4) **uint8_t usdhc_data_t::dataType**
- (5) **size_t usdhc_data_t::blockSize**
- (6) **uint32_t usdhc_data_t::blockCount**
- (7) **uint32_t* usdhc_data_t::rxData**
- (8) **const uint32_t* usdhc_data_t::txData**

42.3.11 struct usdhc_transfer_t**Data Fields**

- **usdhc_data_t * data**
Data to transfer.
- **usdhc_command_t * command**
Command to send.

Field Documentation

- (1) **usdhc_data_t* usdhc_transfer_t::data**
- (2) **usdhc_command_t* usdhc_transfer_t::command**

42.3.12 struct usdhc_transfer_callback_t**Data Fields**

- **void(* CardInserted)(USDHC_Type *base, void *userData)**
Card inserted occurs when DAT3/CD pin is for card detect.
- **void(* CardRemoved)(USDHC_Type *base, void *userData)**
Card removed occurs.
- **void(* SdioInterrupt)(USDHC_Type *base, void *userData)**
SDIO card interrupt occurs.
- **void(* BlockGap)(USDHC_Type *base, void *userData)**
stopped at block gap event
- **void(* TransferComplete)(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)**
Transfer complete callback.
- **void(* ReTuning)(USDHC_Type *base, void *userData)**

Handle the re-tuning.

Field Documentation

- (1) `void(* usdhc_transfer_callback_t::TransferComplete)(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)`
- (2) `void(* usdhc_transfer_callback_t::ReTuning)(USDHC_Type *base, void *userData)`

42.3.13 struct _usdhc_handle

USDHC handle typedef.

Defines the structure to save the USDHC state information and callback function.

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

Data Fields

- `usdhc_data_t *volatile data`
Transfer parameter.
- `usdhc_command_t *volatile command`
Transfer parameter.
- `volatile uint32_t transferredWords`
Transfer status.
- `usdhc_transfer_callback_t callback`
Callback function.
- `void * userData`
Parameter for transfer complete callback.

Field Documentation

- (1) `usdhc_data_t* volatile usdhc_handle_t::data`
Data to transfer.
- (2) `usdhc_command_t* volatile usdhc_handle_t::command`
Command to send.

- (3) `volatile uint32_t usdhc_handle_t::transferredWords`

Words transferred by DATAPORT way.

- (4) usdhc_transfer_callback_t usdhc_handle_t::callback
- (5) void* usdhc_handle_t::userData

42.3.14 struct usdhc_host_t

Data Fields

- USDHC_Type * base
USDHC peripheral base address.
- uint32_t sourceClock_Hz
USDHC source clock frequency united in Hz.
- usdhc_config_t config
USDHC configuration.
- usdhc_capability_t capability
USDHC capability information.
- usdhc_transfer_function_t transfer
USDHC transfer function.

Field Documentation

- (1) USDHC_Type* usdhc_host_t::base
- (2) uint32_t usdhc_host_t::sourceClock_Hz
- (3) usdhc_config_t usdhc_host_t::config
- (4) usdhc_capability_t usdhc_host_t::capability
- (5) usdhc_transfer_function_t usdhc_host_t::transfer

42.4 Macro Definition Documentation

42.4.1 #define FSL_USDHC_DRIVER_VERSION (MAKE_VERSION(2U, 8U, 4U))

42.4.2 #define USDHC_ADMA1_ADDRESS_ALIGN (4096U)

42.4.3 #define USDHC_ADMA1_LENGTH_ALIGN (4096U)

42.4.4 #define USDHC_ADMA2_ADDRESS_ALIGN (4U)

42.4.5 #define USDHC_ADMA2_LENGTH_ALIGN (4U)

42.4.6 #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (12U)

Address/page field	Reserved	6p(*8-*3)*6/8 lightgrayAttribute	04	03	02	01
31 12	11 6	05				
address or data length	000000	Act2	Act1	0	Int	E

ADMA1 descriptor table

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don't care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

ADMA2 action

42.4.7 #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (0xFFFFFU)

42.4.8 #define USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (12U)

42.4.9 #define USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (0xFFFFU)

42.4.10 #define USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK + 1U - 4096U)

Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

42.4.11 #define USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (16U)

Address field	Length	Reserved	6p(*9-*4)*6/9 lightgrayAttribute	04	03	02	01
63 32	31 16	15 06	05				
32-bit address	16-bit length	0000000000	Act2	Act1	0	Int	E

ADMA2 descriptor table

Act2	Act1	Comment	Operation
0	0	No op	Don't care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

ADMA2 action**42.6 Enumeration Type Documentation****42.6.1 anonymous enum**

USDHC status.

Enumerator

kStatus_USDHC_BusyTransferring Transfer is on-going.

kStatus_USDHC_PrepAdmaDescriptorFailed Set DMA descriptor failed.

kStatus_USDHC_SendCommandFailed Send command failed.

kStatus_USDHC_TransferDataFailed Transfer data failed.

kStatus_USDHC_DMADataAddrNotAlign Data address not aligned.

kStatus_USDHC_ReTuningRequest Re-tuning request.

kStatus_USDHC_TuningError Tuning error.

kStatus_USDHC_NotSupport Not support.

kStatus_USDHC_TransferDataComplete Transfer data complete.

kStatus_USDHC_SendCommandSuccess Transfer command complete.

kStatus_USDHC_TransferDMAComplete Transfer DMA complete.

42.6.2 anonymous enum

Host controller capabilities flag mask.

Enumerator

kUSDHC_SupportAdmaFlag Support ADMA.

kUSDHC_SupportHighSpeedFlag Support high-speed.

kUSDHC_SupportDmaFlag Support DMA.

kUSDHC_SupportSuspendResumeFlag Support suspend/resume.

kUSDHC_SupportV330Flag Support voltage 3.3V.

- kUSDHC_SupportV300Flag* Support voltage 3.0V.
- kUSDHC_SupportV180Flag* Support voltage 1.8V.
- kUSDHC_Support4BitFlag* Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.
- kUSDHC_Support8BitFlag* Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.
- kUSDHC_SupportDDR50Flag* SD version 3.0 new feature, supporting DDR50 mode.
- kUSDHC_SupportSDR104Flag* Support SDR104 mode.
- kUSDHC_SupportSDR50Flag* Support SDR50 mode.

42.6.3 anonymous enum

Wakeup event mask.

Enumerator

- kUSDHC_WakeupEventOnCardInt* Wakeup on card interrupt.
- kUSDHC_WakeupEventOnCardInsert* Wakeup on card insertion.
- kUSDHC_WakeupEventOnCardRemove* Wakeup on card removal.
- kUSDHC_WakeupEventsAll* All wakeup events.

42.6.4 anonymous enum

Reset type mask.

Enumerator

- kUSDHC_ResetAll* Reset all except card detection.
- kUSDHC_ResetCommand* Reset command line.
- kUSDHC_ResetData* Reset data line.
- kUSDHC_ResetTuning* Reset tuning circuit.
- kUSDHC_ResetsAll* All reset types.

42.6.5 anonymous enum

Transfer flag mask.

Enumerator

- kUSDHC_EnableDmaFlag* Enable DMA.
- kUSDHC_CommandTypeSuspendFlag* Suspend command.
- kUSDHC_CommandTypeResumeFlag* Resume command.
- kUSDHC_CommandTypeAbortFlag* Abort command.
- kUSDHC_EnableBlockCountFlag* Enable block count.
- kUSDHC_EnableAutoCommand12Flag* Enable auto CMD12.
- kUSDHC_DataReadFlag* Enable data read.

kUSDHC_MultipleBlockFlag Multiple block data read/write.
kUSDHC_EnableAutoCommand23Flag Enable auto CMD23.
kUSDHC_ResponseLength136Flag 136-bit response length.
kUSDHC_ResponseLength48Flag 48-bit response length.
kUSDHC_ResponseLength48BusyFlag 48-bit response length with busy status.
kUSDHC_EnableCrcCheckFlag Enable CRC check.
kUSDHC_EnableIndexCheckFlag Enable index check.
kUSDHC_DataPresentFlag Data present flag.

42.6.6 anonymous enum

Present status flag mask.

Enumerator

kUSDHC_CommandInhibitFlag Command inhibit.
kUSDHC_DataInhibitFlag Data inhibit.
kUSDHC_DataLineActiveFlag Data line active.
kUSDHC_SdClockStableFlag SD bus clock stable.
kUSDHC_WriteTransferActiveFlag Write transfer active.
kUSDHC_ReadTransferActiveFlag Read transfer active.
kUSDHC_BufferWriteEnableFlag Buffer write enable.
kUSDHC_BufferReadEnableFlag Buffer read enable.
kUSDHC_ReTuningRequestFlag Re-tuning request flag, only used for SDR104 mode.
kUSDHC_DelaySettingFinishedFlag Delay setting finished flag.
kUSDHC_CardInsertedFlag Card inserted.
kUSDHC_CommandLineLevelFlag Command line signal level.
kUSDHC_Data0LineLevelFlag Data0 line signal level.
kUSDHC_Data1LineLevelFlag Data1 line signal level.
kUSDHC_Data2LineLevelFlag Data2 line signal level.
kUSDHC_Data3LineLevelFlag Data3 line signal level.
kUSDHC_Data4LineLevelFlag Data4 line signal level.
kUSDHC_Data5LineLevelFlag Data5 line signal level.
kUSDHC_Data6LineLevelFlag Data6 line signal level.
kUSDHC_Data7LineLevelFlag Data7 line signal level.

42.6.7 anonymous enum

Interrupt status flag mask.

Enumerator

kUSDHC_CommandCompleteFlag Command complete.
kUSDHC_DataCompleteFlag Data complete.

kUSDHC_BlockGapEventFlag Block gap event.
kUSDHC_DmaCompleteFlag DMA interrupt.
kUSDHC_BufferWriteReadyFlag Buffer write ready.
kUSDHC_BufferReadReadyFlag Buffer read ready.
kUSDHC_CardInsertionFlag Card inserted.
kUSDHC_CardRemovalFlag Card removed.
kUSDHC_CardInterruptFlag Card interrupt.
kUSDHC_ReTuningEventFlag Re-Tuning event, only for SD3.0 SDR104 mode.
kUSDHC_TuningPassFlag SDR104 mode tuning pass flag.
kUSDHC_TuningErrorFlag SDR104 tuning error flag.
kUSDHC_CommandTimeoutFlag Command timeout error.
kUSDHC_CommandCrcErrorFlag Command CRC error.
kUSDHC_CommandEndBitErrorFlag Command end bit error.
kUSDHC_CommandIndexErrorFlag Command index error.
kUSDHC_DataTimeoutFlag Data timeout error.
kUSDHC_DataCrcErrorFlag Data CRC error.
kUSDHC_DataEndBitErrorFlag Data end bit error.
kUSDHC_AutoCommand12ErrorFlag Auto CMD12 error.
kUSDHC_DmaErrorFlag DMA error.
kUSDHC_CommandErrorFlag Command error.
kUSDHC_DataErrorFlag Data error.
kUSDHC_ErrorFlag All error.
kUSDHC_DataFlag Data interrupts.
kUSDHC_DataDMAFlag Data interrupts.
kUSDHC_CommandFlag Command interrupts.
kUSDHC_CardDetectFlag Card detection interrupts.
kUSDHC_SDR104TuningFlag SDR104 tuning flag.
kUSDHC_AllInterruptFlags All flags mask.

42.6.8 anonymous enum

Auto CMD12 error status flag mask.

Enumerator

kUSDHC_AutoCommand12NotExecutedFlag Not executed error.
kUSDHC_AutoCommand12TimeoutFlag Timeout error.
kUSDHC_AutoCommand12EndBitErrorFlag End bit error.
kUSDHC_AutoCommand12CrcErrorFlag CRC error.
kUSDHC_AutoCommand12IndexErrorFlag Index error.
kUSDHC_AutoCommand12NotIssuedFlag Not issued error.

42.6.9 anonymous enum

Standard tuning flag.

Enumerator

kUSDHC_ExecuteTuning Used to start tuning procedure.

kUSDHC_TuningSampleClockSel When `std_tuning_en` bit is set, this bit is used to select sampleing clock.

42.6.10 anonymous enum

ADMA error status flag mask.

Enumerator

kUSDHC_AdmaLengthMismatchFlag Length mismatch error.

kUSDHC_AdmaDescriptorErrorFlag Descriptor error.

42.6.11 anonymous enum

ADMA error state.

This state is the detail state when ADMA error has occurred.

Enumerator

kUSDHC_AdmaErrorStateStopDma Stop DMA, previous location set in the ADMA system address is errored address.

kUSDHC_AdmaErrorStateFetchDescriptor Fetch descriptor, current location set in the ADMA system address is errored address.

kUSDHC_AdmaErrorStateChangeAddress Change address, no DMA error has occurred.

kUSDHC_AdmaErrorStateTransferData Transfer data, previous location set in the ADMA system address is errored address.

kUSDHC_AdmaErrorStateInvalidLength Invalid length in ADMA descriptor.

kUSDHC_AdmaErrorStateInvalidDescriptor Invalid descriptor fetched by ADMA.

kUSDHC_AdmaErrorState ADMA error state.

42.6.12 anonymous enum

Force event bit position.

Enumerator

kUSDHC_ForceEventAutoCommand12NotExecuted Auto CMD12 not executed error.

kUSDHC_ForceEventAutoCommand12Timeout Auto CMD12 timeout error.
kUSDHC_ForceEventAutoCommand12CrcError Auto CMD12 CRC error.
kUSDHC_ForceEventEndBitError Auto CMD12 end bit error.
kUSDHC_ForceEventAutoCommand12IndexError Auto CMD12 index error.
kUSDHC_ForceEventAutoCommand12NotIssued Auto CMD12 not issued error.
kUSDHC_ForceEventCommandTimeout Command timeout error.
kUSDHC_ForceEventCommandCrcError Command CRC error.
kUSDHC_ForceEventCommandEndBitError Command end bit error.
kUSDHC_ForceEventCommandIndexError Command index error.
kUSDHC_ForceEventDataTimeout Data timeout error.
kUSDHC_ForceEventDataCrcError Data CRC error.
kUSDHC_ForceEventDataEndBitError Data end bit error.
kUSDHC_ForceEventAutoCommand12Error Auto CMD12 error.
kUSDHC_ForceEventCardInt Card interrupt.
kUSDHC_ForceEventDmaError Dma error.
kUSDHC_ForceEventTuningError Tuning error.
kUSDHC_ForceEventsAll All force event flags mask.

42.6.13 enum usdhc_transfer_direction_t

Enumerator

kUSDHC_TransferDirectionReceive USDHC transfer direction receive.
kUSDHC_TransferDirectionSend USDHC transfer direction send.

42.6.14 enum usdhc_data_bus_width_t

Enumerator

kUSDHC_DataBusWidth1Bit 1-bit mode
kUSDHC_DataBusWidth4Bit 4-bit mode
kUSDHC_DataBusWidth8Bit 8-bit mode

42.6.15 enum usdhc_endian_mode_t

Enumerator

kUSDHC_EndianModeBig Big endian mode.
kUSDHC_EndianModeHalfWordBig Half word big endian mode.
kUSDHC_EndianModeLittle Little endian mode.

42.6.16 enum usdhc_dma_mode_t

Enumerator

- kUSDHC_DmaModeSimple* External DMA.
- kUSDHC_DmaModeAdma1* ADMA1 is selected.
- kUSDHC_DmaModeAdma2* ADMA2 is selected.
- kUSDHC_ExternalDMA* External DMA mode selected.

42.6.17 anonymous enum

SDIO control flag mask.

Enumerator

- kUSDHC_StopAtBlockGapFlag* Stop at block gap.
- kUSDHC_ReadWaitControlFlag* Read wait control.
- kUSDHC_InterruptAtBlockGapFlag* Interrupt at block gap.
- kUSDHC_ReadDoneNo8CLK* Read done without 8 clk for block gap.
- kUSDHC_ExactBlockNumberReadFlag* Exact block number read.

42.6.18 enum usdhc_boot_mode_t

Enumerator

- kUSDHC_BootModeNormal* Normal boot.
- kUSDHC_BootModeAlternative* Alternative boot.

42.6.19 enum usdhc_card_command_type_t

Enumerator

- kCARD_CommandTypeNormal* Normal command.
- kCARD_CommandTypeSuspend* Suspend command.
- kCARD_CommandTypeResume* Resume command.
- kCARD_CommandTypeAbort* Abort command.
- kCARD_CommandTypeEmpty* Empty command.

42.6.20 enum usdhc_card_response_type_t

Defines the command response type from card to host controller.

Enumerator

- kCARD_ResponseNone* Response type: none.
- kCARD_ResponseR1* Response type: R1.
- kCARD_ResponseR1b* Response type: R1b.
- kCARD_ResponseR2* Response type: R2.
- kCARD_ResponseR3* Response type: R3.
- kCARD_ResponseR4* Response type: R4.
- kCARD_ResponseR5* Response type: R5.
- kCARD_ResponseR5b* Response type: R5b.
- kCARD_ResponseR6* Response type: R6.
- kCARD_ResponseR7* Response type: R7.

42.6.21 anonymous enum

The mask for the control/status field in ADMA1 descriptor.

Enumerator

- kUSDHC_Adma1DescriptorValidFlag* Valid flag.
- kUSDHC_Adma1DescriptorEndFlag* End flag.
- kUSDHC_Adma1DescriptorInterruptFlag* Interrupt flag.
- kUSDHC_Adma1DescriptorActivity1Flag* Activity 1 flag.
- kUSDHC_Adma1DescriptorActivity2Flag* Activity 2 flag.
- kUSDHC_Adma1DescriptorTypeNop* No operation.
- kUSDHC_Adma1DescriptorTypeTransfer* Transfer data.
- kUSDHC_Adma1DescriptorTypeLink* Link descriptor.
- kUSDHC_Adma1DescriptorTypeSetLength* Set data length.

42.6.22 anonymous enum

ADMA1 descriptor control and status mask.

Enumerator

- kUSDHC_Adma2DescriptorValidFlag* Valid flag.
- kUSDHC_Adma2DescriptorEndFlag* End flag.
- kUSDHC_Adma2DescriptorInterruptFlag* Interrupt flag.
- kUSDHC_Adma2DescriptorActivity1Flag* Activity 1 mask.
- kUSDHC_Adma2DescriptorActivity2Flag* Activity 2 mask.
- kUSDHC_Adma2DescriptorTypeNop* No operation.
- kUSDHC_Adma2DescriptorTypeReserved* Reserved.
- kUSDHC_Adma2DescriptorTypeTransfer* Transfer type.
- kUSDHC_Adma2DescriptorTypeLink* Link type.

42.6.23 anonymous enum

ADMA descriptor configuration flag.

Enumerator

kUSDHC_AdmaDescriptorSingleFlag Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

kUSDHC_AdmaDescriptorMultipleFlag Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

42.6.24 enum usdhc_burst_len_t

Enumerator

kUSDHC_EnBurstLenForINCR Enable burst len for INCR.

kUSDHC_EnBurstLenForINCR4816 Enable burst len for INCR4/INCR8/INCR16.

kUSDHC_EnBurstLenForINCR4816WRAP Enable burst len for INCR4/8/16 WRAP.

42.6.25 anonymous enum

Transfer data type definition.

Enumerator

kUSDHC_TransferDataNormal Transfer normal read/write data.

kUSDHC_TransferDataTuning Transfer tuning data.

kUSDHC_TransferDataBoot Transfer boot data.

kUSDHC_TransferDataBootcontinuous Transfer boot data continuously.

42.7 Function Documentation

42.7.1 void USDHC_Init (***USDHC_Type * base***, ***const usdhc_config_t * config***)

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

<i>base</i>	USDHC peripheral base address.
<i>config</i>	USDHC configuration information.

Return values

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

42.7.2 void USDHC_Deinit (**USDHC_Type** * *base*)

Parameters

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

42.7.3 bool USDHC_Reset (**USDHC_Type** * *base*, **uint32_t** *mask*, **uint32_t** *timeout*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The reset type mask(_usdhc_reset).
<i>timeout</i>	Timeout for reset.

Return values

<i>true</i>	Reset successfully.
<i>false</i>	Reset failed.

42.7.4 status_t USDHC_SetAdmaTableConfig (**USDHC_Type** * *base*, **usdhc_adma_config_t** * *dmaConfig*, **usdhc_data_t** * *dataConfig*, **uint32_t** *flags*)

A high level DMA descriptor configuration function.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dmaConfig</i>	ADMA configuration
<i>dataConfig</i>	Data descriptor
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag .

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

42.7.5 status_t USDHC_SetInternalDmaConfig (**USDHC_Type * *base*, **usdhc_adma_config_t** * *dmaConfig*, const **uint32_t** * *dataAddr*, **bool enAutoCmd23**)**

This function is used to config the USDHC DMA related registers.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dmaConfig</i>	ADMA configuration.
<i>dataAddr</i>	Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
<i>enAutoCmd23</i>	Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

42.7.6 status_t USDHC_SetADMA2Descriptor (**uint32_t * *admaTable*, **uint32_t admaTableWords**, const **uint32_t** * *dataBufferAddr*, **uint32_t dataBytes**, **uint32_t flags**)**

Parameters

<i>admaTable</i>	ADMA table address.
<i>admaTableWords</i>	ADMA table length.
<i>dataBufferAddr</i>	Data buffer address.
<i>dataBytes</i>	Data Data length.
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag .

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

42.7.7 **status_t USDHC_SetADMA1Descriptor (uint32_t * *admaTable*, uint32_t *admaTableWords*, const uint32_t * *dataBufferAddr*, uint32_t *dataBytes*, uint32_t *flags*)**

Parameters

<i>admaTable</i>	ADMA table address.
<i>admaTableWords</i>	ADMA table length.
<i>dataBufferAddr</i>	Data buffer address.
<i>dataBytes</i>	Data length.
<i>flags</i>	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag .

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

42.7.8 **static void USDHC_EnableInternalDMA (USDHC_Type * *base*, bool *enable*) [inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable or disable flag

42.7.9 static void USDHC_EnableInterruptStatus (**USDHC_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	Interrupt status flags mask(_usdhc_interrupt_status_flag).

42.7.10 static void USDHC_DisableInterruptStatus (**USDHC_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

42.7.11 static void USDHC_EnableInterruptSignal (**USDHC_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

42.7.12 static void USDHC_DisableInterruptSignal (**USDHC_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

42.7.13 static uint32_t USDHC_GetEnabledInterruptStatusFlags (**USDHC_Type** * *base*) [inline], [static]

Parameters

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

Returns

Current interrupt status flags mask([_usdhc_interrupt_status_flag](#)).

42.7.14 static uint32_t USDHC_GetInterruptStatusFlags (**USDHC_Type** * *base*) [inline], [static]

Parameters

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

Returns

Current interrupt status flags mask([_usdhc_interrupt_status_flag](#)).

42.7.15 static void USDHC_ClearInterruptStatusFlags (**USDHC_Type** * *base*, uint32_t *mask*) [inline], [static]

write 1 clears

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The interrupt status flags mask(_usdhc_interrupt_status_flag).

42.7.16 static uint32_t USDHC_GetAutoCommand12ErrorStatusFlags (**USDHC_Type** * *base*) [inline], [static]

Parameters

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

Returns

Auto command 12 error status flags mask([_usdhc_auto_command12_error_status_flag](#)).

42.7.17 static uint32_t USDHC_GetAdmaErrorStatusFlags (**USDHC_Type** * *base*) [inline], [static]

Parameters

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

Returns

ADMA error status flags mask([_usdhc_adma_error_status_flag](#)).

42.7.18 static uint32_t USDHC_GetPresentStatusFlags (**USDHC_Type** * *base*) [inline], [static]

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

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

Returns

Present USDHC's status flags mask([_usdhc_present_status_flag](#)).

42.7.19 **void USDHC_GetCapability (**USDHC_Type** * *base*, **usdhc_capability_t** * *capability*)**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>capability</i>	Structure to save capability information.

42.7.20 static void USDHC_ForceClockOn (**USDHC_Type * *base*, **bool** *enable*)
[inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

42.7.21 **uint32_t USDHC_SetSdClock (**USDHC_Type** * *base*, **uint32_t** *srcClock_Hz*,
uint32_t *busClock_Hz*)**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>srcClock_Hz</i>	USDHC source clock frequency united in Hz.
<i>busClock_Hz</i>	SD bus clock frequency united in Hz.

Returns

The nearest frequency of busClock_Hz configured for SD bus.

42.7.22 **bool USDHC_SetCardActive (**USDHC_Type** * *base*, **uint32_t** *timeout*)**

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>timeout</i>	Timeout to initialize card.

Return values

<i>true</i>	Set card active successfully.
<i>false</i>	Set card active failed.

42.7.23 static void USDHC_AassertHardwareReset (**USDHC_Type** * *base*, **bool** *high*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>high</i>	1 or 0 level

42.7.24 static void USDHC_SetDataBusWidth (**USDHC_Type** * *base*, **usdhc_data_bus_width_t** *width*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>width</i>	Data transfer width.

42.7.25 static void USDHC_WriteData (**USDHC_Type** * *base*, **uint32_t** *data*) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>data</i>	The data about to be sent.

**42.7.26 static uint32_t USDHC_ReadData (USDHC_Type * *base*) [inline],
[static]**

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

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

Returns

The data has been read.

42.7.27 void USDHC_SendCommand (**USDHC_Type** * *base*, **usdhc_command_t** * *command*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>command</i>	configuration

42.7.28 static void USDHC_EnableWakeupEvent (**USDHC_Type** * *base*, **uint32_t** *mask*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	Wakeup events mask(_usdhc_wakeup_event).
<i>enable</i>	True to enable, false to disable.

42.7.29 static void USDHC_CardDetectByData3 (**USDHC_Type** * *base*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

42.7.30 static bool USDHC_DetectCardInsert (**USDHC_Type** * *base*) [inline], [static]

Parameters

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

42.7.31 static void USDHC_EnableSdioControl (**USDHC_Type** * *base*, **uint32_t** *mask*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	SDIO card control flags mask(_usdhc_sdio_control_flag).
<i>enable</i>	True to enable, false to disable.

42.7.32 static void USDHC_SetContinueRequest (**USDHC_Type** * *base*) [inline], [static]

Parameters

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

42.7.33 static void USDHC_RequestStopAtBlockGap (**USDHC_Type** * *base*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	True to stop at block gap, false to normal transfer.

42.7.34 void USDHC_SetMmcBootConfig (**USDHC_Type** * *base*, **const** **usdhc_boot_config_t** * *config*)

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
```

```
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

<i>base</i>	USDHC peripheral base address.
<i>config</i>	The MMC boot configuration information.

42.7.35 static void USDHC_EnableMmcBoot (**USDHC_Type** * *base*, **bool** *enable*) [**inline**], [**static**]

Parameters

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

42.7.36 static void USDHC_SetForceEvent (**USDHC_Type** * *base*, **uint32_t** *mask*) [**inline**], [**static**]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>mask</i>	The force events bit position (_usdhc_force_event).

42.7.37 static void UDSHC_SelectVoltage (**USDHC_Type** * *base*, **bool** *en18v*) [**inline**], [**static**]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>en18v</i>	True means 1.8V, false means 3.0V.

42.7.38 static bool USDHC_RequestTuningForSDR50 (**USDHC_Type** * *base*) [**inline**], [**static**]

When this bit set, application shall perform tuning for SDR50 mode.

Parameters

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

42.7.39 static bool USDHC_RequestReTuning (**USDHC_Type** * *base*) [inline], [static]

When this bit is set, user should do manual tuning or standard tuning function.

Parameters

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

42.7.40 static void USDHC_EnableAutoTuning (**USDHC_Type** * *base*, **bool** *enable*) [inline], [static]

This function should be called after tuning function execute pass, auto tuning will handle by hardware.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

42.7.41 void USDHC_EnableAutoTuningForCmdAndData (**USDHC_Type** * *base*)

Parameters

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

42.7.42 void USDHC_EnableManualTuning (**USDHC_Type** * *base*, **bool** *enable*)

User should handle the tuning cmd and find the boundary of the delay then calculate a average value which will be configured to the **CLK_TUNE_CTRL_STATUS** This function should be called before function [USDHC_AdjustDelayForManualTuning](#).

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	tuning enable flag

42.7.43 static uint32_t USDHC_GetTuningDelayStatus (**USDHC_Type * *base*)
[inline], [static]**

Parameters

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

Return values

<i>CLK</i>	Tuning Control and Status register value.
------------	---

42.7.44 status_t USDHC_SetTuningDelay (**USDHC_Type * *base*, uint32_t *preDelay*,
uint32_t *outDelay*, uint32_t *postDelay*)**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>preDelay</i>	Set the number of delay cells on the feedback clock between the feedback clock and CLK_PRE.
<i>outDelay</i>	Set the number of delay cells on the feedback clock between CLK_PRE and CLK_OUT.
<i>postDelay</i>	Set the number of delay cells on the feedback clock between CLK_OUT and CLK_POST.

Return values

<i>kStatus_Fail</i>	config the delay setting fail
<i>kStatus_Success</i>	config the delay setting success

42.7.45 status_t USDHC_AdjustDelayForManualTuning (**USDHC_Type * *base*,
uint32_t *delay*)**

Deprecated Do not use this function. It has been superceded by USDHC_SetTuingDelay

Parameters

<i>base</i>	USDHC peripheral base address.
<i>delay</i>	setting configuration

Return values

<i>kStatus_Fail</i>	config the delay setting fail
<i>kStatus_Success</i>	config the delay setting success

42.7.46 static void USDHC_SetStandardTuningCounter (**USDHC_Type** * *base*, **uint8_t** *counter*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>counter</i>	tuning counter

Return values

<i>kStatus_Fail</i>	config the delay setting fail
<i>kStatus_Success</i>	config the delay setting success

42.7.47 void USDHC_EnableStandardTuning (**USDHC_Type** * *base*, **uint32_t** *tuningStartTap*, **uint32_t** *step*, **bool** *enable*)

The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>tuningStartTap</i>	start tap
<i>step</i>	tuning step

<i>enable</i>	enable/disable flag
---------------	---------------------

42.7.48 static uint32_t USDHC_GetExecuteStdTuningStatus (USDHC_Type * *base*) [inline], [static]

Parameters

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

42.7.49 static uint32_t USDHC_CheckStdTuningResult (USDHC_Type * *base*) [inline], [static]

Parameters

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

42.7.50 static uint32_t USDHC_CheckTuningError (USDHC_Type * *base*) [inline], [static]

Parameters

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

42.7.51 void USDHC_EnableDDRMode (USDHC_Type * *base*, bool *enable*, uint32_t *nibblePos*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag
<i>nibblePos</i>	nibble position

42.7.52 **static void USDHC_EnableHS400Mode (USDHC_Type * *base*, bool *enable*) [inline], [static]**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

42.7.53 static void USDHC_ResetStrobeDLL (USDHC_Type * *base*) [inline], [static]

Parameters

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

42.7.54 static void USDHC_EnableStrobeDLL (USDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>enable</i>	enable/disable flag

42.7.55 void USDHC_ConfigStrobeDLL (USDHC_Type * *base*, uint32_t *delayTarget*, uint32_t *updateInterval*)

Parameters

<i>base</i>	USDHC peripheral base address.
<i>delayTarget</i>	delay target
<i>updateInterval</i>	update interval

42.7.56 static void USDHC_SetStrobeDIIOverride (USDHC_Type * *base*, uint32_t *delayTaps*) [inline], [static]

Parameters

<i>base</i>	USDHC peripheral base address.
<i>delayTaps</i>	Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

42.7.57 static uint32_t USDHC_GetStrobeDLLStatus (**USDHC_Type * *base*)
[inline], [static]**

Parameters

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

42.7.58 void USDHC_SetDataConfig (**USDHC_Type * *base*, **usdhc_transfer_direction_t** *dataDirection*, **uint32_t** *blockCount*, **uint32_t** *blockSize*)**

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dataDirection</i>	Data direction, tx or rx.
<i>blockCount</i>	Data block count.
<i>blockSize</i>	Data block size.

42.7.59 void USDHC_TransferCreateHandle (**USDHC_Type * *base*,
usdhc_handle_t * *handle*, **const usdhc_transfer_callback_t** * *callback*,
void * *userData*)**

Parameters

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

<i>callback</i>	Structure pointer to contain all callback functions.
<i>userData</i>	Callback function parameter.

42.7.60 status_t USDHC_TransferNonBlocking (***USDHC_Type * base,*** ***usdhc_handle_t * handle,*** ***usdhc_adma_config_t * dmaConfig,*** ***usdhc_transfer_t * transfer*)**

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

Call API [USDHC_TransferCreateHandle](#) when calling this API.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle.
<i>dmaConfig</i>	ADMA configuration.
<i>transfer</i>	Transfer content.

Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_USDHC_Busy-Transferring</i>	Busy transferring.
<i>kStatus_USDHC_-PrepareAdmaDescriptor-Failed</i>	Prepare ADMA descriptor failed.
<i>kStatus_Success</i>	Operate successfully.

42.7.61 status_t USDHC_TransferBlocking (***USDHC_Type * base,*** ***usdhc_adma_config_t * dmaConfig,*** ***usdhc_transfer_t * transfer*)**

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note

There is no need to call API [USDHC_TransferCreateHandle](#) when calling this API.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>dmaConfig</i>	adma configuration
<i>transfer</i>	Transfer content.

Return values

<i>kStatus_InvalidArgument</i>	Argument is invalid.
<i>kStatus_USDHC_PrepAdmaDescriptorFailed</i>	Prepare ADMA descriptor failed.
<i>kStatus_USDHC_SendCommandFailed</i>	Send command failed.
<i>kStatus_USDHC_TransferDataFailed</i>	Transfer data failed.
<i>kStatus_Success</i>	Operate successfully.

42.7.62 void USDHC_TransferHandleIRQ (**USDHC_Type** * *base*, **usdhc_handle_t** * *handle*)

This function deals with the IRQs on the given host controller.

Parameters

<i>base</i>	USDHC peripheral base address.
<i>handle</i>	USDHC handle.

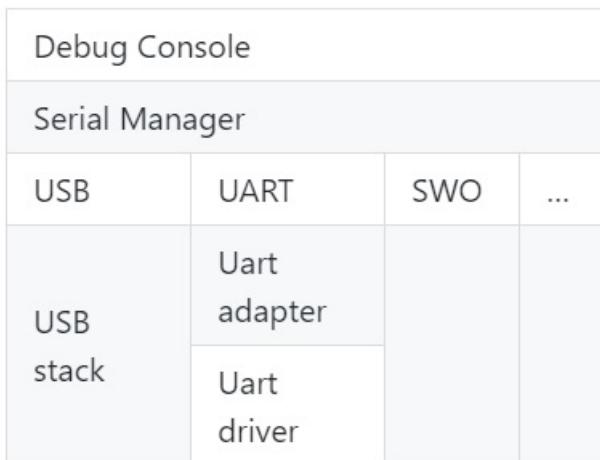
Chapter 43

Debug Console

43.1 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

43.2 Function groups

43.2.1 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);
```

43.2.2 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 */

```

43.2.3 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_- REDIRECT_TO_SDK</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_SDK</code>	undefined	Low level peripheral*	semihost
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	undefined	semihost	semihost
<code>DEBUGCONSOLE_- DISABLE</code>	defined	No output	Low level peripheral
<code>DEBUGCONSOLE_- DISABLE</code>	undefined	No output	semihost

SDK_DEBUGCONSOLE	SDK_DEBUGCONSOLE_UART	PRINTF	printf
-------------------------	------------------------------	---------------	---------------

* the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

43.3 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.

43.4 Macro Definition Documentation

43.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

43.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

43.4.3 #define DEBUGCONSOLE_DISABLE 2U

43.4.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

43.4.5 #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.

43.5 Function Documentation

43.5.1 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.

Returns

<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"> • kSerialPort_Uart, • kSerialPort_UsbCdc
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully
------------------------	------------------------

43.5.2 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.

43.5.3 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.

43.5.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

43.5.5 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.

43.5.6 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.

43.5.7 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.

43.5.8 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.

43.5.9 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.

43.5.10 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.

43.5.11 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.

43.5.12 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.

43.5.13 **status_t DbgConsole_TryGetchar (char * *ch*)**

Parameters

<i>ch</i>	the address of char to receive
-----------	--------------------------------

Returns

Indicates get char was successful or not.

43.6 debug console configuration

The configuration is used for debug console only.

43.6.1 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.

43.6.2 Macro Definition Documentation

43.6.2.1 #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.

43.6.2.2 #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.

43.6.2.3 #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.

43.6.2.4 #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.

43.6.2.5 #define DEBUG_CONSOLE_SCANF_MAX_LOG_LEN (20U)

As same as the DEBUG_CONSOLE_BUFFER_PRINTF_MAX_LOG_LEN.

43.6.2.6 #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

43.6.2.7 #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.

43.6.2.8 #define BOARD_USE_VIRTUALCOM 0U

43.7 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.

43.7.1 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.

43.7.2 Guide Semihosting for Keil µVision

NOTE: Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

43.7.3 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.

43.7.4 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.

43.8 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.

43.8.1 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

43.8.1.1 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.

43.8.2 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.

43.8.3 Guide SWO for MCUXpresso IDE

NOTE: MCUX support SWO for LPC-Link2 debug probe only.

43.8.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 44

Notification Framework

44.1 Overview

This section describes the programming interface of the Notifier driver.

44.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.
The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{
    status_t ret = kStatus_Success;

    ...
    ...

    return ret;
}
// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *
    userData)
```

```

{
    ...
    ...
    ...
}

...
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
        kNOTIFIER_CallbackBeforeAfter,
        (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
        APP_PowerModeSwitch, NULL);
    ...

    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
        kNOTIFIER_PolicyAgreement);
}

```

Data Structures

- struct [notifier_notification_block_t](#)
notification block passed to the registered callback function. [More...](#)
- struct [notifier_callback_config_t](#)
Callback configuration structure. [More...](#)
- struct [notifier_handle_t](#)
Notifier handle structure. [More...](#)

Typedefs

- [typedef void notifier_user_config_t](#)
Notifier user configuration type.
- [typedef status_t\(* notifier_user_function_t \)\(notifier_user_config_t *targetConfig, void *userData\)](#)

- *Notifier user function prototype Use this function to execute specific operations in configuration switch.*
typedef status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)
Callback prototype.

Enumerations

- **enum _notifier_status {**
kStatus_NOTIFIER_ErrorNotificationBefore,
kStatus_NOTIFIER_ErrorNotificationAfter }
Notifier error codes.
- **enum notifier_policy_t {**
kNOTIFIER_PolicyAgreement,
kNOTIFIER_PolicyForcible }
Notifier policies.
- **enum notifier_notification_type_t {**
kNOTIFIER_NotifyRecover = 0x00U,
kNOTIFIER_NotifyBefore = 0x01U,
kNOTIFIER_NotifyAfter = 0x02U }
Notification type.
- **enum notifier_callback_type_t {**
kNOTIFIER_CallbackBefore = 0x01U,
kNOTIFIER_CallbackAfter = 0x02U,
kNOTIFIER_CallbackBeforeAfter = 0x03U }
The callback type, which indicates kinds of notification the callback handles.

Functions

- **status_t NOTIFIER_CreateHandle (notifier_handle_t *notifierHandle, notifier_user_config_t **configs, uint8_t configsNumber, notifier_callback_config_t *callbacks, uint8_t callbacksNumber, notifier_user_function_t userFunction, void *userData)**
Creates a Notifier handle.
- **status_t NOTIFIER_SwitchConfig (notifier_handle_t *notifierHandle, uint8_t configIndex, notifier_policy_t policy)**
Switches the configuration according to a pre-defined structure.
- **uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t *notifierHandle)**
This function returns the last failed notification callback.

44.3 Data Structure Documentation

44.3.1 struct notifier_notification_block_t

Data Fields

- **notifier_user_config_t * targetConfig**
Pointer to target configuration.
- **notifier_policy_t policy**
Configure transition policy.
- **notifier_notification_type_t notifyType**

Configure notification type.

Field Documentation

- (1) **notifier_user_config_t* notifier_notification_block_t::targetConfig**
- (2) **notifier_policy_t notifier_notification_block_t::policy**
- (3) **notifier_notification_type_t notifier_notification_block_t::notifyType**

44.3.2 struct notifier_callback_config_t

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback.

Data Fields

- **notifier_callback_t callback**
Pointer to the callback function.
- **notifier_callback_type_t callbackType**
Callback type.
- **void *callbackData**
Pointer to the data passed to the callback.

Field Documentation

- (1) **notifier_callback_t notifier_callback_config_t::callback**
- (2) **notifier_callback_type_t notifier_callback_config_t::callbackType**
- (3) **void* notifier_callback_config_t::callbackData**

44.3.3 struct notifier_handle_t

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. [NOTIFIER_CreateHandle\(\)](#) must be called to initialize this handle.

Data Fields

- **notifier_user_config_t ** configsTable**
Pointer to configure table.
- **uint8_t configsNumber**
Number of configurations.

- `notifier_callback_config_t * callbacksTable`
Pointer to callback table.
- `uint8_t callbacksNumber`
Maximum number of callback configurations.
- `uint8_t errorCallbackIndex`
Index of callback returns error.
- `uint8_t currentConfigIndex`
Index of current configuration.
- `notifier_user_function_t userFunction`
User function.
- `void * userData`
User data passed to user function.

Field Documentation

- (1) `notifier_user_config_t** notifier_handle_t::configsTable`
- (2) `uint8_t notifier_handle_t::configsNumber`
- (3) `notifier_callback_config_t* notifier_handle_t::callbacksTable`
- (4) `uint8_t notifier_handle_t::callbacksNumber`
- (5) `uint8_t notifier_handle_t::errorCallbackIndex`
- (6) `uint8_t notifier_handle_t::currentConfigIndex`
- (7) `notifier_user_function_t notifier_handle_t::userFunction`
- (8) `void* notifier_handle_t::userData`

44.4 Typedef Documentation

44.4.1 `typedef void notifier_user_config_t`

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

44.4.2 `typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)`

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, `NOTIFIER_SwitchConfig()` exits.

Parameters

<i>targetConfig</i>	target Configuration.
<i>userData</i>	Refers to other specific data passed to user function.

Returns

An error code or kStatus_Success.

44.4.3 **typedef status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)**

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the [notifier_callback_config_t](#) callback configuration structure. Depending on callback type, function of this prototype is called (see [NOTIFIER_SwitchConfig\(\)](#)) before configuration switch, after it or in both use cases to notify about the switch progress (see [notifier_callback_type_t](#)). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see [notifier_notification_block_t](#)) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see [notifier_policy_t](#)), the callback may deny the execution of the user function by returning an error code different than kStatus_Success (see [NOTIFIER_SwitchConfig\(\)](#)).

Parameters

<i>notify</i>	Notification block.
<i>data</i>	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or kStatus_Success.

44.5 Enumeration Type Documentation

44.5.1 enum _notifier_status

Used as return value of Notifier functions.

Enumerator

kStatus_NOTIFIER_ErrorNotificationBefore An error occurs during send "BEFORE" notification.

kStatus_NOTIFIER_ErrorNotificationAfter An error occurs during send "AFTER" notification.

44.5.2 enum notifier_policy_t

Defines whether the user function execution is forced or not. For `kNOTIFIER_PolicyForcible`, the user function is executed regardless of the callback results, while `kNOTIFIER_PolicyAgreement` policy is used to exit `NOTIFIER_SwitchConfig()` when any of the callbacks returns error code. See also `NOTIFIER_SwitchConfig()` description.

Enumerator

kNOTIFIER_PolicyAgreement `NOTIFIER_SwitchConfig()` method is exited when any of the callbacks returns error code.

kNOTIFIER_PolicyForcible The user function is executed regardless of the results.

44.5.3 enum notifier_notification_type_t

Used to notify registered callbacks

Enumerator

kNOTIFIER_NotifyRecover Notify IP to recover to previous work state.

kNOTIFIER_NotifyBefore Notify IP that configuration setting is going to change.

kNOTIFIER_NotifyAfter Notify IP that configuration setting has been changed.

44.5.4 enum notifier_callback_type_t

Used in the callback configuration structure (`notifier_callback_config_t`) to specify when the registered callback is called during configuration switch initiated by the `NOTIFIER_SwitchConfig()`. Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect `NOTIFIER_SwitchConfig()` execution. See the `NOTIFIER_SwitchConfig()` and `notifier_policy_t` documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

kNOTIFIER_CallbackBefore Callback handles BEFORE notification.

kNOTIFIER_CallbackAfter Callback handles AFTER notification.

kNOTIFIER_CallbackBeforeAfter Callback handles BEFORE and AFTER notification.

44.6 Function Documentation

44.6.1 `status_t NOTIFIER_CreateHandle(notifier_handle_t * notifierHandle,
notifier_user_config_t ** configs, uint8_t configsNumber, notifier_callback-
_config_t * callbacks, uint8_t callbacksNumber, notifier_user_function_t
userFunction, void * userData)`

Parameters

<i>notifierHandle</i>	A pointer to the notifier handle.
<i>configs</i>	A pointer to an array with references to all configurations which is handled by the Notifier.
<i>configsNumber</i>	Number of configurations. Size of the configuration array.
<i>callbacks</i>	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
<i>callbacks-Number</i>	Number of registered callbacks. Size of the callbacks array.
<i>userFunction</i>	User function.
<i>userData</i>	User data passed to user function.

Returns

An error Code or kStatus_Success.

44.6.2 **status_t NOTIFIER_SwitchConfig (notifier_handle_t * *notifierHandle*, uint8_t *configIndex*, notifier_policy_t *policy*)**

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER_SwitchConfig() exits.

Parameters

<i>notifierHandle</i>	pointer to notifier handle
<i>configIndex</i>	Index of the target configuration.
<i>policy</i>	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus_Success.

44.6.3 `uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t *notifierHandle)`

This function returns an index of the last callback that failed during the configuration switch while the last [NOTIFIER_SwitchConfig\(\)](#) was called. If the last [NOTIFIER_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

<i>notifierHandle</i>	Pointer to the notifier handle
-----------------------	--------------------------------

Returns

Callback Index of the last failed callback or value equal to callbacks count.

Chapter 45

Shell

45.1 Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

45.2 Function groups

45.2.1 Initialization

To initialize the Shell middleware, call the `SHELL_Init()` function with these parameters. This function automatically enables the middleware.

```
shell_status_t SHELL_Init(shell_handle_t shellHandle,  
    serial_handle_t serialHandle, char *prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

This example shows how to call the `SHELL_Init()` given the user configuration structure.

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
```

45.2.2 Advanced Feature

- Support to get a character from standard input devices.

```
static shell_status_t SHELL_GetChar(shell_context_handle_t *shellContextHandle, uint8_t *ch);
```

Commands	Description
help	List all the registered commands.
exit	Exit program.

45.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");  
SHELL_Task(s_shellHandle);
```

Data Structures

- struct `shell_command_t`
User command data configuration structure. More...

Macros

- #define `SHELL_NON_BLOCKING_MODE` SERIAL_MANAGER_NON_BLOCKING_MODE
Whether use non-blocking mode.
- #define `SHELL_AUTO_COMPLETE` (1U)
Macro to set on/off auto-complete feature.
- #define `SHELL_BUFFER_SIZE` (64U)
Macro to set console buffer size.
- #define `SHELL_MAX_ARGS` (8U)
Macro to set maximum arguments in command.
- #define `SHELL_HISTORY_COUNT` (3U)
Macro to set maximum count of history commands.
- #define `SHELL_IGNORE_PARAMETER_COUNT` (0xFF)
Macro to bypass arguments check.
- #define `SHELL_HANDLE_SIZE`
The handle size of the shell module.
- #define `SHELL_USE_COMMON_TASK` (0U)
Macro to determine whether use common task.
- #define `SHELL_TASK_PRIORITY` (2U)
Macro to set shell task priority.
- #define `SHELL_TASK_STACK_SIZE` (1000U)
Macro to set shell task stack size.
- #define `SHELL_HANDLE_DEFINE`(name) uint32_t name[((`SHELL_HANDLE_SIZE` + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the shell handle.
- #define `SHELL_COMMAND_DEFINE`(command, descriptor, callback, paramInt)
Defines the shell command structure.
- #define `SHELL_COMMAND`(command) &g_shellCommand##command
Gets the shell command pointer.

Typedefs

- typedef void * `shell_handle_t`
The handle of the shell module.
- typedef `shell_status_t`(* `cmd_function_t`)(`shell_handle_t` shellHandle, int32_t argc, char **argv)
User command function prototype.

Enumerations

- enum `shell_status_t` {

`kStatus_SHELL_Success` = kStatus_Success,

`kStatus_SHELL_Error` = MAKE_STATUS(kStatusGroup_SHELL, 1),

`kStatus_SHELL_OpenWriteHandleFailed` = MAKE_STATUS(kStatusGroup_SHELL, 2),

`kStatus_SHELL_OpenReadHandleFailed` = MAKE_STATUS(kStatusGroup_SHELL, 3),

`kStatus_SHELL_RetUsage` = MAKE_STATUS(kStatusGroup_SHELL, 4) }

Shell status.

Shell functional operation

- `shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char *prompt)`
Initializes the shell module.
- `shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t *shellCommand)`
Registers the shell command.
- `shell_status_t SHELL_UnregisterCommand (shell_command_t *shellCommand)`
Unregisters the shell command.
- `shell_status_t SHELL_Write (shell_handle_t shellHandle, const char *buffer, uint32_t length)`
Sends data to the shell output stream.
- `int SHELL_Printf (shell_handle_t shellHandle, const char *formatString,...)`
Writes formatted output to the shell output stream.
- `shell_status_t SHELL_WriteSynchronization (shell_handle_t shellHandle, const char *buffer, uint32_t length)`
Sends data to the shell output stream with OS synchronization.
- `int SHELL_PrintfSynchronization (shell_handle_t shellHandle, const char *formatString,...)`
Writes formatted output to the shell output stream with OS synchronization.
- `void SHELL_ChangePrompt (shell_handle_t shellHandle, char *prompt)`
Change shell prompt.
- `void SHELL_PrintPrompt (shell_handle_t shellHandle)`
Print shell prompt.
- `void SHELL_Task (shell_handle_t shellHandle)`
The task function for Shell.
- `static bool SHELL_checkRunningInIsr (void)`
Check if code is running in ISR.

45.3 Data Structure Documentation

45.3.1 struct shell_command_t

Data Fields

- `const char * pcCommand`
The command that is executed.
- `char * pcHelpString`
String that describes how to use the command.
- `const cmd_function_t pFuncCallBack`
A pointer to the callback function that returns the output generated by the command.
- `uint8_t cExpectedNumberOfParameters`
Commands expect a fixed number of parameters, which may be zero.
- `list_element_t link`
link of the element

Field Documentation

(1) `const char* shell_command_t::pcCommand`

For example "help". It must be all lower case.

(2) `char* shell_command_t::pcHelpString`

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

(3) `const cmd_function_t shell_command_t::pFuncCallBack`**(4) `uint8_t shell_command_t::cExpectedNumberOfParameters`**

45.4 Macro Definition Documentation

45.4.1 `#define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE`

45.4.2 `#define SHELL_AUTO_COMPLETE (1U)`

45.4.3 `#define SHELL_BUFFER_SIZE (64U)`

45.4.4 `#define SHELL_MAX_ARGS (8U)`

45.4.5 `#define SHELL_HISTORY_COUNT (3U)`

45.4.6 `#define SHELL_HANDLE_SIZE`

Value:

```
(160U + SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE +
    SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + \
    SERIAL_MANAGER_WRITE_HANDLE_SIZE)
```

It is the sum of the SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE

45.4.7 #define SHELL_USE_COMMON_TASK (0U)

45.4.8 #define SHELL_TASK_PRIORITY (2U)

45.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

**45.4.10 #define SHELL_HANDLE_DEFINE(*name*) uint32_t
name[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]**

This macro is used to define a 4 byte aligned shell handle. Then use "(shell_handle_t)*name*" to get the shell handle.

The macro should be global and could be optional. You could also define shell handle by yourself.

This is an example,

```
* SHELL_HANDLE_DEFINE(shellHandle);
*
```

Parameters

<i>name</i>	The name string of the shell handle.
-------------	--------------------------------------

**45.4.11 #define SHELL_COMMAND_DEFINE(*command*, *descriptor*, *callback*,
paramCount)**

Value:

```
\shell_command_t g_shellCommand##command = {
    (#command), (descriptor), (callback), (paramCount), {0},      \
}
```

This macro is used to define the shell command structure [shell_command_t](#). And then uses the macro SHELL_COMMAND to get the command structure pointer. The macro should not be used in any function.

This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
<i>descriptor</i>	The description of the command is used for showing the command usage when "help" is typing.
<i>callback</i>	The callback of the command is used to handle the command line when the input command is matched.
<i>paramCount</i>	The max parameter count of the current command.

45.4.12 #define SHELL_COMMAND(*command*) &g_shellCommand##*command*

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
----------------	--

45.5 Typedef Documentation

45.5.1 **typedef shell_status_t(* cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char **argv)**

45.6 Enumeration Type Documentation

45.6.1 enum shell_status_t

Enumerator

kStatus_SHELL_Success Success.

kStatus_SHELL_Error Failed.

kStatus_SHELL_OpenWriteHandleFailed Open write handle failed.

kStatus_SHELL_OpenReadHandleFailed Open read handle failed.

kStatus_SHELL_RetUsage RetUsage for print cmd usage.

45.7 Function Documentation

45.7.1 **shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char * prompt)**

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL-

_Init function by passing in these parameters. This is an example.

```
* static SHELL_HANDLE_DEFINE(s_shellHandle);
* SHELL_Init((shell_handle_t)s_shellHandle,
*             (serial_handle_t)s_serialHandle, "Test@SHELL>");
```

Parameters

<i>shellHandle</i>	Pointer to point to a memory space of size SHELL_HANDLE_SIZE 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: SHELL_HANDLE_DEFINE(shellHandle) ; or <code>uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>serialHandle</i>	The serial manager module handle pointer.
<i>prompt</i>	The string prompt pointer of Shell. Only the global variable can be passed.

Return values

<i>kStatus_SHELL_Success</i>	The shell initialization succeed.
<i>kStatus_SHELL_Error</i>	An error occurred when the shell is initialized.
<i>kStatus_SHELL_Open-WriteHandleFailed</i>	Open the write handle failed.
<i>kStatus_SHELL_Open-ReadHandleFailed</i>	Open the read handle failed.

45.7.2 **shell_status_t SHELL_RegisterCommand (shell_handle_t *shellHandle*, shell_command_t * *shellCommand*)**

This function is used to register the shell command by using the command configuration `shell_command_config_t`. This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>shellCommand</i>	The command element.

Return values

<i>kStatus_SHELL_Success</i>	Successfully register the command.
<i>kStatus_SHELL_Error</i>	An error occurred.

45.7.3 **shell_status_t SHELL_UnregisterCommand (shell_command_t * *shellCommand*)**

This function is used to unregister the shell command.

Parameters

<i>shellCommand</i>	The command element.
---------------------	----------------------

Return values

<i>kStatus_SHELL_Success</i>	Successfully unregister the command.
------------------------------	--------------------------------------

45.7.4 **shell_status_t SHELL_Write (shell_handle_t *shellHandle*, const char * *buffer*, uint32_t *length*)**

This function is used to send data to the shell output stream.

Parameters

<i>shellHandle</i>	The shell 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_SHELL_Success</i>	Successfully send data.
------------------------------	-------------------------

<i>kStatus_SHELL_Error</i>	An error occurred.
----------------------------	--------------------

45.7.5 int SHELL_Printf (shell_handle_t *shellHandle*, const char * *formatString*, ...)

Call this function to write a formatted output to the shell output stream.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>formatString</i>	Format string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

45.7.6 shell_status_t SHELL_WriteSynchronization (shell_handle_t *shellHandle*, const char * *buffer*, uint32_t *length*)

This function is used to send data to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell 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_SHELL_Success</i>	Successfully send data.
<i>kStatus_SHELL_Error</i>	An error occurred.

45.7.7 int SHELL_PrintfSynchronization (shell_handle_t *shellHandle*, const char * *formatString*, ...)

Call this function to write a formatted output to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>formatString</i>	Format string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

45.7.8 void SHELL_ChangePrompt (shell_handle_t *shellHandle*, char * *prompt*)

Call this function to change shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>prompt</i>	The string which will be used for command prompt

Returns

NULL.

45.7.9 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

Returns

NULL.

45.7.10 void SHELL_Task (shell_handle_t *shellHandle*)

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

45.7.11 static bool SHELL_checkRunningInIsr(void) [inline], [static]

This function is used to check if code running in ISR.

Return values

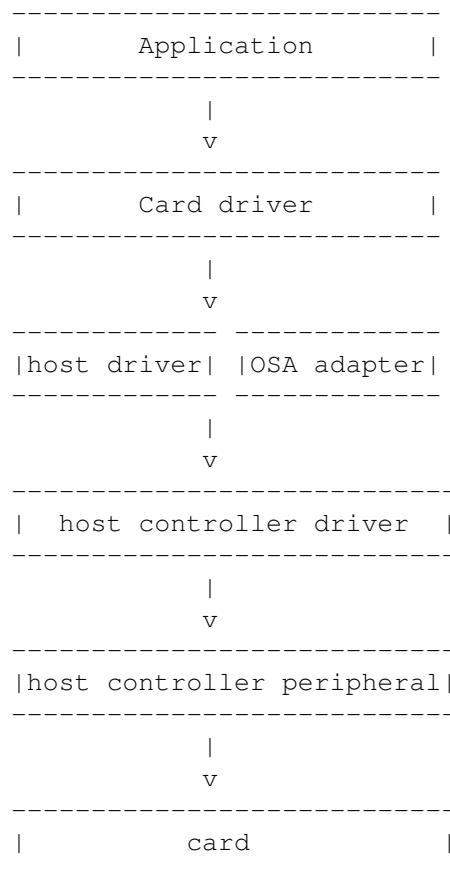
<i>TRUE</i>	if code runing in ISR.
-------------	------------------------

Chapter 46

Cards: Secure Digital Card/Embedded MultiMedia Card/SD-IO Card

46.1 Overview

The MCUXpresso SDK provides drivers to access the Secure Digital Card(up to v3.0), Embedded Multi-Media Card(up to v5.0) and sdio card(up to v3.0) based on the SDHC/USDHC/SDIF driver. Here is a simple block diagram about the drivers:



Modules

- [MMC Card Driver](#)
- [SD Card Driver](#)
- [SDIO Card Driver](#)
- [SDMMC Common](#)
- [SDMMC HOST Driver](#)
- [SDMMC OSA](#)

46.2 SDIO Card Driver

46.2.1 Overview

The SDIO card driver provide card initialization/IO direct and extend command interface.

46.2.2 SDIO CARD Operation

error log support

Not supported yet.

User configurable

Board dependency

Mutual exclusive access support for RTOS

SDIO driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
SDIO_Deinit(card); /* This function will destroy the created mutex */
SDIO_Init(card);
```

Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/sdmmc_examples/

Data Structures

- struct [sdio_card_t](#)
SDIO card state. [More...](#)

Macros

- #define [FSL_SDIO_DRIVER_VERSION](#) (MAKE_VERSION(2U, 4U, 1U)) /*2.4.1*/
Middleware version.
- #define [FSL_SDIO_MAX_IO_NUMS](#) (7U)
sdio device support maximum IO number

Typedefs

- `typedef void(* sdio_io_irq_handler_t)(sdio_card_t *card, uint32_t func)`
sdio io handler

Enumerations

- `enum sdio_io_direction_t {
 kSDIO_IORead = 0U,
 kSDIO_IOWrite = 1U }`
sdio io read/write direction

Initialization and deinitialization

- `status_t SDIO_Init (sdio_card_t *card)`
SDIO card init function.
- `void SDIO_Deinit (sdio_card_t *card)`
SDIO card_deinit, include card and host_deinit.
- `status_t SDIO_CardInit (sdio_card_t *card)`
Initializes the card.
- `void SDIO_CardDeinit (sdio_card_t *card)`
Deinitializes the card.
- `status_t SDIO_HostInit (sdio_card_t *card)`
initialize the host.
- `void SDIO_HostDeinit (sdio_card_t *card)`
Deinitializes the host.
- `void SDIO_HostDoReset (sdio_card_t *card)`
reset the host.
- `void SDIO_SetCardPower (sdio_card_t *card, bool enable)`
set card power.
- `status_t SDIO_CardInActive (sdio_card_t *card)`
set SDIO card to inactive state
- `status_t SDIO_GetCardCapability (sdio_card_t *card, sdio_func_num_t func)`
get SDIO card capability
- `status_t SDIO_SetBlockSize (sdio_card_t *card, sdio_func_num_t func, uint32_t blockSize)`
set SDIO card block size
- `status_t SDIO_CardReset (sdio_card_t *card)`
set SDIO card reset
- `status_t SDIO_SetDataBusWidth (sdio_card_t *card, sdio_bus_width_t busWidth)`
set SDIO card data bus width
- `status_t SDIO_SwitchToHighSpeed (sdio_card_t *card)`
switch the card to high speed
- `status_t SDIO_ReadCIS (sdio_card_t *card, sdio_func_num_t func, const uint32_t *tupleList, uint32_t tupleNum)`
read SDIO card CIS for each function
- `status_t SDIO_PollingCardInsert (sdio_card_t *card, uint32_t status)`
sdio wait card detect function.
- `bool SDIO_IsCardPresent (sdio_card_t *card)`

sdio card present check function.

IO operations

- `status_t SDIO_IO_Write_Direct` (`sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *data, bool raw`)

IO direct write transfer function.
- `status_t SDIO_IO_Read_Direct` (`sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *data`)

IO direct read transfer function.
- `status_t SDIO_IO_RW_Direct` (`sdio_card_t *card, sdio_io_direction_t direction, sdio_func_num_t func, uint32_t regAddr, uint8_t dataIn, uint8_t *dataOut`)

IO direct read/write transfer function.
- `status_t SDIO_IO_Write_Extended` (`sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *buffer, uint32_t count, uint32_t flags`)

IO extended write transfer function.
- `status_t SDIO_IO_Read_Extended` (`sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *buffer, uint32_t count, uint32_t flags`)

IO extended read transfer function.
- `status_t SDIO_EnableIOInterrupt` (`sdio_card_t *card, sdio_func_num_t func, bool enable`)

enable IO interrupt
- `status_t SDIO_EnableIO` (`sdio_card_t *card, sdio_func_num_t func, bool enable`)

enable IO and wait IO ready
- `status_t SDIO_SelectIO` (`sdio_card_t *card, sdio_func_num_t func`)

select IO
- `status_t SDIO_AbortIO` (`sdio_card_t *card, sdio_func_num_t func`)

Abort IO transfer.
- `status_t SDIO_SetDriverStrength` (`sdio_card_t *card, sd_driver_strength_t driverStrength`)

Set driver strength.
- `status_t SDIO_EnableAsyncInterrupt` (`sdio_card_t *card, bool enable`)

Enable/Disable Async interrupt.
- `status_t SDIO_GetPendingInterrupt` (`sdio_card_t *card, uint8_t *pendingInt`)

Get pending interrupt.
- `status_t SDIO_IO_Transfer` (`sdio_card_t *card, sdio_command_t cmd, uint32_t argument, uint32_t blockSize, uint8_t *txData, uint8_t *rxData, uint16_t dataSize, uint32_t *response`)

sdio card io transfer function.
- `void SDIO_SetIOIRQHandler` (`sdio_card_t *card, sdio_func_num_t func, sdio_io_irq_handler_t handler`)

sdio set io IRQ handler.
- `status_t SDIO_HandlePendingIOInterrupt` (`sdio_card_t *card`)

sdio card io pending interrupt handle function.

46.2.3 Data Structure Documentation

46.2.3.1 struct _sdio_card

sdio card descriptor

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- **sdmmchost_t * host**
Host information.
- **sdio_usr_param_t usrParam**
user parameter
- **bool noInternalAlign**
use this flag to disable sdmmc align.
- **uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]**
internal buffer
- **bool isHostReady**
use this flag to indicate if need host re-init or not
- **bool memPresentFlag**
indicate if memory present
- **uint32_t busClock_Hz**
SD bus clock frequency united in Hz.
- **uint32_t relativeAddress**
Relative address of the card.
- **uint8_t sdVersion**
SD version.
- **sd_timing_mode_t currentTiming**
current timing mode
- **sd_driver_strength_t driverStrength**
driver strength
- **sd_max_current_t maxCurrent**
card current limit
- **sdmmc_operation_voltage_t operationVoltage**
card operation voltage
- **uint8_t sdioVersion**
SDIO version.
- **uint8_t cccrVersioin**
CCCR version.
- **uint8_t ioTotalNumber**
total number of IO function
- **uint32_t cccrflags**
Flags in _sd_card_flag.
- **uint32_t io0blockSize**
record the io0 block size
- **uint32_t ocr**
Raw OCR content, only 24bit available for SDIO card.
- **uint32_t commonCISPointer**
point to common CIS
- **sdio_common_cis_t commonCIS**
CIS table.
- **sdio_fbr_t ioFBR [FSL_SDIO_MAX_IO_NUMS]**
FBR table.
- **sdio_func_cis_t funcCIS [FSL_SDIO_MAX_IO_NUMS]**
function CIS table
- **sdio_io_irq_handler_t ioIRQHandler [FSL_SDIO_MAX_IO_NUMS]**

- *io IRQ handler*
- `uint8_t ioIntIndex`
used to record current enabled io interrupt index
- `uint8_t ioIntNums`
used to record total enabled io interrupt numbers
- `sdmmc_osa_mutex_t lock`
card access lock

Field Documentation

(1) `bool sdio_card_t::noInternalAlign`

If disable, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

46.2.4 Macro Definition Documentation

46.2.4.1 `#define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 1U)) /*2.4.1*/`

46.2.5 Enumeration Type Documentation

46.2.5.1 `enum sdio_io_direction_t`

Enumerator

- `kSDIO_IORead` io read
- `kSDIO_IOWrite` io write

46.2.6 Function Documentation

46.2.6.1 `status_t SDIO_Init(sdio_card_t * card)`

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDIO_Deinit(card);
* SDIO_Init(card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Go-IdleFailed</i>	
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i>	
<i>kStatus_SDMMC_SDIO-InvalidCard</i>	
<i>kStatus_SDMMC_SDIO-InvalidVoltage</i>	
<i>kStatus_SDMMC_Send-RelativeAddressFailed</i>	
<i>kStatus_SDMMC_Select-CardFailed</i>	
<i>kStatus_SDMMC_SDIO-SwitchHighSpeedFail</i>	
<i>kStatus_SDMMC_SDIO-ReadCISFail</i>	
<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.2 void SDIO_Deinit (**sdio_card_t** * *card*)

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.2.6.3 status_t SDIO_CardInit (**sdio_card_t** * *card*)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return *kStatus_SDMMC_HostNotReady*.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDIO_CardDeinit(card);
* SDIO_CardInit(card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_Not-SupportYet</i>	Card not support.
<i>kStatus_SDMMC_Send-OperationCondition-Failed</i>	Send operation condition failed.
<i>kStatus_SDMMC_All-SendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_Send-RelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_Send-CsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_Select-CardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_Send-ScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetBus-WidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-HighSpeedFailed</i>	Switch high speed failed.

<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

46.2.6.4 void SDIO_CardDeinit (*sdio_card_t* * *card*)

This function deinitializes the specific card.

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.2.6.5 status_t SDIO_HostInit (*sdio_card_t* * *card*)

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.2.6.6 void SDIO_HostDeinit (*sdio_card_t* * *card*)

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.2.6.7 void SDIO_HostDoReset (*sdio_card_t* * *card*)

This function reset the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.2.6.8 void SDIO_SetCardPower (*sdio_card_t* * *card*, *bool enable*)

The power off operation depend on host or the user define power on function.

Parameters

<i>card</i>	card descriptor.
<i>enable</i>	true is power on, false is power off.

46.2.6.9 status_t SDIO_CardInActive (**sdio_card_t * card**)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.10 status_t SDIO_GetCardCapability (**sdio_card_t * card, sdio_func_num_t func**)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.11 status_t SDIO_SetBlockSize (**sdio_card_t * card, sdio_func_num_t func, uint32_t blockSize**)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	io number
<i>blockSize</i>	block size

Return values

<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	
<i>kStatus_SDMMC_SDIO_InvalidArgument</i>	
<i>kStatus_Success</i>	

46.2.6.12 status_t SDIO_CardReset (*sdio_card_t * card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.13 status_t SDIO_SetDataBusWidth (*sdio_card_t * card, sdio_bus_width_t busWidth*)

Parameters

<i>card</i>	Card descriptor.
<i>busWidth</i>	bus width

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.14 status_t SDIO_SwitchToHighSpeed (*sdio_card_t * card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_SDMMC_SDIO_-SwitchHighSpeedFail</i>	
<i>kStatus_Success</i>	

46.2.6.15 status_t SDIO_ReadCIS (*sdio_card_t * card*, *sdio_func_num_t func*, *const uint32_t * tupleList*, *uint32_t tupleNum*)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	io number
<i>tupleList</i>	code list
<i>tupleNum</i>	code number

Return values

<i>kStatus_SDMMC_SDIO_-ReadCISFail</i>	
<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.16 status_t SDIO_PollingCardInsert (*sdio_card_t * card*, *uint32_t status*)

Detect card through GPIO, CD, DATA3.

Parameters

<i>card</i>	card descriptor.
<i>status</i>	detect status, kSD_Inserted or kSD_Removed.

46.2.6.17 bool SDIO_IsCardPresent (*sdio_card_t* * *card*)

Parameters

<i>card</i>	card descriptor.
-------------	------------------

46.2.6.18 status_t SDIO_IO_Write_Direct (*sdio_card_t* * *card*, *sdio_func_num_t* *func*, *uint32_t* *regAddr*, *uint8_t* * *data*, *bool* *raw*)

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>data</i>	the data pointer to write
<i>raw</i>	flag, indicate read after write or write only

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.19 status_t SDIO_IO_Read_Direct (*sdio_card_t* * *card*, *sdio_func_num_t* *func*, *uint32_t* *regAddr*, *uint8_t* * *data*)

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>data</i>	pointer to read

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.20 status_t SDIO_IO_RW_Direct (*sdio_card_t * card*, *sdio_io_direction_t direction*, *sdio_func_num_t func*, *uint32_t regAddr*, *uint8_t dataIn*, *uint8_t * dataOut*)

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>direction</i>	io access direction, please reference <i>sdio_io_direction_t</i> .
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>dataIn</i>	data to write
<i>dataOut</i>	data pointer for readback data, support both for read and write, when application want readback the data after write command, <i>dataOut</i> should not be NULL.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.21 status_t SDIO_IO_Write_Extended (*sdio_card_t * card*, *sdio_func_num_t func*, *uint32_t regAddr*, *uint8_t * buffer*, *uint32_t count*, *uint32_t flags*)

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>buffer</i>	data buffer to write
<i>count</i>	data count
<i>flags</i>	write flags

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_SDMMC_SDIO_InvalidArgument</i>	
<i>kStatus_Success</i>	

46.2.6.22 **status_t SDIO_IO_Read_Extended (*sdio_card_t * card*, *sdio_func_num_t func*, *uint32_t regAddr*, *uint8_t * buffer*, *uint32_t count*, *uint32_t flags*)**

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>buffer</i>	data buffer to read
<i>count</i>	data count
<i>flags</i>	write flags

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_SDMMC_SDIO_InvalidArgument</i>	
<i>kStatus_Success</i>	

46.2.6.23 **status_t SDIO_EnableOIInterrupt (*sdio_card_t * card*, *sdio_func_num_t func*, *bool enable*)**

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.24 status_t SDIO_EnableIO (***sdio_card_t * card, sdio_func_num_t func, bool enable***)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.25 status_t SDIO_SelectIO (***sdio_card_t * card, sdio_func_num_t func***)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.26 status_t SDIO_AbortIO (*sdio_card_t * card*, *sdio_func_num_t func*)

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.27 status_t SDIO_SetDriverStrength (**sdio_card_t * card, sd_driver_strength_t driverStrength**)

Parameters

<i>card</i>	Card descriptor.
<i>driverStrength</i>	target driver strength.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.28 status_t SDIO_EnableAsyncInterrupt (**sdio_card_t * card, bool enable**)

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	true is enable, false is disable.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
-------------------------------------	--

<i>kStatus_Success</i>

46.2.6.29 status_t SDIO_GetPendingInterrupt (*sdio_card_t * card, uint8_t * pendingInt*)

Parameters

<i>card</i>	Card descriptor.
<i>pendingInt</i>	pointer store pending interrupt

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

46.2.6.30 status_t SDIO_IO_Transfer (*sdio_card_t * card, sdio_command_t cmd, uint32_t argument, uint32_t blockSize, uint8_t * txData, uint8_t * rxData, uint16_t dataSize, uint32_t * response*)

This function can be used for transfer direct/extend command. Please pay attention to the non-align data buffer address transfer, if data buffer address can not meet host controller internal DMA requirement, sdio driver will try to use internal align buffer if data size is not bigger than internal buffer size, Align address transfer always can get a better performance, so if application want sdio driver make sure buffer address align,

Please note it is a thread safe function.

Parameters

<i>card</i>	card descriptor.
<i>cmd</i>	command to transfer
<i>argument</i>	argument to transfer
<i>blockSize</i>	used for block mode.
<i>txData</i>	tx buffer pointer or NULL

<i>rxData</i>	rx buffer pointer or NULL
<i>dataSize</i>	transfer data size
<i>response</i>	reponse pointer, if application want read response back, please set it to a NON-NULL pointer.

46.2.6.31 void SDIO_SetIOIRQHandler (*sdio_card_t* * *card*, *sdio_func_num_t* *func*, *sdio_io_irq_handler_t* *handler*)

Parameters

<i>card</i>	card descriptor.
<i>func</i>	function io number.
<i>handler</i>	io IRQ handler.

46.2.6.32 status_t SDIO_HandlePendingIOInterrupt (*sdio_card_t* * *card*)

This function is used to handle the pending io interrupt. To reigster a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, true);
* SDIO_SetIOIRQHandler(card, 0, func0_handler);
*
```

call it in interrupt callback

```
* SDIO_HandlePendingIOInterrupt(card);
*
```

To releae a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, false);
* SDIO_SetIOIRQHandler(card, 0, NULL);
*
```

Parameters

<i>card</i>	card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

46.3 SD Card Driver

46.3.1 Overview

The SDCARD driver provide card initialization/read/write/erase interface.

46.3.2 SD CARD Operation

error log support

Lots of error log has been added to sd relate functions, if error occurs during initial/read/write, please enable the error log print functionality with `#define SDMMC_ENABLE_LOG_PRINT 1` And rerun the project then user can check what kind of error happened.

User configurable

```
typedef struct _sd_card
{
    sdmmchost_t *host;
    sd_usr_param_t usrParam;
    bool isHostReady;
    bool noInternalAlign;
    uint32_t busClock_Hz;
    uint32_t relativeAddress;
    uint32_t version;
    uint32_t flags;
    uint8_t internalBuffer[FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE];
    uint32_t ocr;
    sd_cid_t cid;
    sd_csd_t csd;
    sd_scr_t scr;
    sd_status_t stat;
    uint32_t blockCount;
    uint32_t blockSize;
    sd_timing_mode_t currentTiming;
    sd_driver_strength_t driverStrength;
    sd_max_current_t maxCurrent;
    sdmmc_operation_voltage_t operationVoltage;
    sdmmc_osa_mutex_t lock;
} sd_card_t;
```

Part of The variables above is user configurable,

1. host

Application need to provide host controller base address and the host's source clock frequency, etc.
For example:

```
/* allocate dma descriptor buffer for host controller */
s_host.dmaDesBuffer
s_host.dmaDesBufferWordsNum
/* */
((sd_card_t *)card)->host
((sd_card_t *)card)->host->hostController.base
((sd_card_t *)card)->host->hostController.sourceClock_Hz = BOARD_USDHC1ClockConfiguration();

/* allocate resource for sdmmc osa layer */
((sd_card_t *)card)->host->hostEvent = &s_event;
```

2. sdcard_usr_param_t usrParam

```
/* board layer configuration register */
((sd_card_t *)card)->usrParam.cd      = &s_cd;
((sd_card_t *)card)->usrParam.pwr      = BOARD_SDCardPowerControl;
((sd_card_t *)card)->usrParam.ioStrength = BOARD_SD_Pin_Config;
((sd_card_t *)card)->usrParam.ioVoltage  = &s_ioVoltage;
((sd_card_t *)card)->usrParam.maxFreq    = BOARD_SDMMC_SD_HOST_SUPPORT_SDR104_FREQ;

a. cd-which allow application define the card insert/remove callback function, redefine the card detect
   timeout ms and also allow application determine how to detect card.
b. pwr-which allow application redefine the card power on/off function.
c. ioStrength-which is used to switch the signal pin configurations include driver strength/speed mode
   dynamically for different timing(SDR/HS timing) mode, reference the function defined sdmmc_config.c
d. ioVoltage-which allow application register io voltage switch function instead of using the function host
   driver provided for SDR/HS200/HS400 timing.
e. maxFreq-which allow application set the maximum bus clock that the board support.
```

1. bool noInternalAlign

Sdmmc include an address align internal buffer(to use host controller internal DMA), to improve read/write performance while application cannot make sure the data address used to read/write is align, set it to true will achieve a better performance.

2. sd_timing_mode_t currentTiming

It is used to indicate the currentTiming the card is working on, however sdmmc also support preset timing mode, then sdmmc will try to switch to this timing first, if failed, a valid timing will switch to automatically. Generally, user may not set this variable if you don't know what kind of timing the card support, sdmmc will switch to the highest timing which the card support.

3. sd_driver_strength_t driverStrength

Choose a valid card driver strength if application required and call SD_SetDriverStrength in application.

4. sd_max_current_t maxCurrent

Choose a valid card current if application required and call SD_SetMaxCurrent in application.

Mutual exclusive access support for RTOS

SDCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
SD_Deinit(card); /* This function will destroy the created mutex */
SD_Init(card);
```

Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/sdmmc_examples/

Data Structures

- struct `sd_card_t`
SD card state. [More...](#)

Macros

- #define **FSL_SD_DRIVER_VERSION** (MAKE_VERSION(2U, 4U, 2U)) /*2.4.2*/
 Driver version.

Enumerations

- enum {
 kSD_SupportHighCapacityFlag = (1U << 1U),
 kSD_Support4BitWidthFlag = (1U << 2U),
 kSD_SupportSdhcFlag = (1U << 3U),
 kSD_SupportSdxcFlag = (1U << 4U),
 kSD_SupportVoltage180v = (1U << 5U),
 kSD_SupportSetBlockCountCmd = (1U << 6U),
 kSD_SupportSpeedClassControlCmd = (1U << 7U) }
- SD card flags.*

SDCARD Function

- **status_t SD_Init (sd_card_t *card)**
Initializes the card on a specific host controller.
- **void SD_Deinit (sd_card_t *card)**
Deinitializes the card.
- **status_t SD_CardInit (sd_card_t *card)**
Initializes the card.
- **void SD_CardDeinit (sd_card_t *card)**
Deinitializes the card.
- **status_t SD_HostInit (sd_card_t *card)**
initialize the host.
- **void SD_HostDeinit (sd_card_t *card)**
Deinitializes the host.
- **void SD_HostDoReset (sd_card_t *card)**
reset the host.
- **void SD_SetCardPower (sd_card_t *card, bool enable)**
set card power.
- **status_t SD_PollingCardInsert (sd_card_t *card, uint32_t status)**
sd wait card detect function.
- **bool SD_IsCardPresent (sd_card_t *card)**
sd card present check function.
- **bool SD_CheckReadOnly (sd_card_t *card)**
Checks whether the card is write-protected.
- **status_t SD_SelectCard (sd_card_t *card, bool isSelected)**
Send SELECT_CARD command to set the card to be transfer state or not.
- **status_t SD_ReadStatus (sd_card_t *card)**
Send ACMD13 to get the card current status.
- **status_t SD_ReadBlocks (sd_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)**

- *Reads blocks from the specific card.*
status_t SD_WriteBlocks (*sd_card_t* *card, const *uint8_t* *buffer, *uint32_t* startBlock, *uint32_t* blockCount)
- *Writes blocks of data to the specific card.*
status_t SD_EraseBlocks (*sd_card_t* *card, *uint32_t* startBlock, *uint32_t* blockCount)
- *Erases blocks of the specific card.*
status_t SD_SetDriverStrength (*sd_card_t* *card, *sd_driver_strength_t* driverStrength)
select card driver strength select card driver strength
- *select max current select max operation current*
status_t SD_SetMaxCurrent (*sd_card_t* *card, *sd_max_current_t* maxCurrent)
- *Polling card idle status.*
status_t SD_PollingCardStatusBusy (*sd_card_t* *card, *uint32_t* timeoutMs)

46.3.3 Data Structure Documentation

46.3.3.1 struct sd_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- **sdmmchost_t * host**
Host configuration.
- **sd_usr_param_t usrParam**
user parameter
- **bool isHostReady**
use this flag to indicate if need host re-init or not
- **bool noInternalAlign**
used to enable/disable the functionality of the exchange buffer
- **uint32_t busClock_Hz**
SD bus clock frequency unitized in Hz.
- **uint32_t relativeAddress**
Relative address of the card.
- **uint32_t version**
Card version.
- **uint32_t flags**
Flags in _sd_card_flag.
- **uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]**
internal buffer
- **uint32_t ocr**
Raw OCR content.
- **sd_cid_t cid**
CID.
- **sd_csd_t csd**
CSD.
- **sd_scr_t scr**
SCR.
- **sd_status_t stat**

- *sd 512 bit status*
- **uint32_t blockCount**
Card total block number.
- **uint32_t blockSize**
Card block size.
- **sd_timing_mode_t currentTiming**
current timing mode
- **sd_driver_strength_t driverStrength**
driver strength
- **sd_max_current_t maxCurrent**
card current limit
- **sdmmc_operation_voltage_t operationVoltage**
card operation voltage
- **sdmmc_osa_mutex_t lock**
card access lock

46.3.4 Macro Definition Documentation

46.3.4.1 #define FSL_SD_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 2U)) /*2.4.2*/

46.3.5 Enumeration Type Documentation

46.3.5.1 anonymous enum

Enumerator

kSD_SupportHighCapacityFlag Support high capacity.

kSD_Support4BitWidthFlag Support 4-bit data width.

kSD_SupportSdhcFlag Card is SDHC.

kSD_SupportSdxcFlag Card is SDXC.

kSD_SupportVoltage180v card support 1.8v voltage

kSD_SupportSetBlockCountCmd card support cmd23 flag

kSD_SupportSpeedClassControlCmd card support speed class control flag

46.3.6 Function Documentation

46.3.6.1 status_t SD_Init (sd_card_t * card)

This function initializes the card on a specific host controller, it is consist of host init, card detect, card init function, however user can ignore this high level function, instead of use the low level function, such as SD_CardInit, SD_HostInit, SD_CardDetect.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

* `SD_Deinit(card);`

```
* SD_Init(card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_HostNotReady</i>	host is not ready.
<i>kStatus_SDMMC_GoIdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_NotSupportYet</i>	Card not support.
<i>kStatus_SDMMC_HandShakeOperationConditionFailed</i>	Send operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_SendRelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_SendScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_SwitchBusTimingFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Set card block size failed.

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

46.3.6.2 void SD_Deinit (*sd_card_t* * *card*)

This function deinitializes the specific card and host. Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.3.6.3 *status_t* SD_CardInit (*sd_card_t* * *card*)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return *kStatus_SDMMC_HostNotReady*.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SD_CardDeinit(card);
* SD_CardInit(card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_Not-SupportYet</i>	Card not support.
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i>	Send operation condition failed.

<i>kStatus_SDMMC_AllSendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_SendRelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_SendScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_SwitchBusTimingFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

46.3.6.4 void SD_CardDeinit(*sd_card_t* * *card*)

This function deinitializes the specific card. Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.3.6.5 status_t SD_HostInit(*sd_card_t* * *card*)

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.3.6.6 void SD_HostDeinit(*sd_card_t* * *card*)

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.3.6.7 void SD_HostDoReset (*sd_card_t* * *card*)

This function reset the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.3.6.8 void SD_SetCardPower (*sd_card_t* * *card*, *bool* *enable*)

The power off operation depend on host or the user define power on function.

Parameters

<i>card</i>	card descriptor.
<i>enable</i>	true is power on, false is power off.

46.3.6.9 *status_t* SD_PollingCardInsert (*sd_card_t* * *card*, *uint32_t* *status*)

Detect card through GPIO, CD, DATA3.

Parameters

<i>card</i>	card descriptor.
<i>status</i>	detect status, kSD_Inserted or kSD_Removed.

46.3.6.10 *bool* SD_IsCardPresent (*sd_card_t* * *card*)

Parameters

<i>card</i>	card descriptor.
-------------	------------------

46.3.6.11 *bool* SD_CheckReadOnly (*sd_card_t* * *card*)

This function checks if the card is write-protected via the CSD register.

Parameters

<i>card</i>	The specific card.
-------------	--------------------

Return values

<i>true</i>	Card is read only.
<i>false</i>	Card isn't read only.

46.3.6.12 status_t SD_SelectCard (*sd_card_t * card, bool isSelected*)

Parameters

<i>card</i>	Card descriptor.
<i>isSelected</i>	True to set the card into transfer state, false to disselect.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

46.3.6.13 status_t SD_ReadStatus (*sd_card_t * card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_SendApplicationCommandFailed</i>	send application command failed.

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

46.3.6.14 status_t SD_ReadBlocks (*sd_card_t * card*, *uint8_t * buffer*, *uint32_t startBlock*, *uint32_t blockCount*)

This function reads blocks from the specific card with default block size defined by the SDHC_CARD_DEFAULT_BLOCK_SIZE.

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save the data read from card.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to read.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

46.3.6.15 status_t SD_WriteBlocks (*sd_card_t * card*, *const uint8_t * buffer*, *uint32_t startBlock*, *uint32_t blockCount*)

This function writes blocks to the specific card with default block size 512 bytes.

Please note,

1. It is a thread safe function.
2. It is a async write function which means that the card status may still busy after the function return.

Application can call function SD_PollingCardStatusBusy to wait card status idle after the write operation.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer holding the data to be written to the card.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to write.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

46.3.6.16 **status_t SD_EraseBlocks (sd_card_t * card, uint32_t startBlock, uint32_t blockCount)**

This function erases blocks of the specific card with default block size 512 bytes.

Please note,

1. It is a thread safe function.
2. It is a async erase function which means that the card status may still busy after the function return.
Application can call function SD_PollingCardStatusBusy to wait card status idle after the erase operation.

Parameters

<i>card</i>	Card descriptor.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to erase.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Send status failed.
<i>kStatus_Success</i>	Operate successfully.

46.3.6.17 **status_t SD_SetDriverStrength (*sd_card_t * card*, *sd_driver_strength_t driverStrength*)**

Parameters

<i>card</i>	Card descriptor.
<i>driverStrength</i>	Driver strength

46.3.6.18 **status_t SD_SetMaxCurrent (*sd_card_t * card*, *sd_max_current_t maxCurrent*)**

Parameters

<i>card</i>	Card descriptor.
<i>maxCurrent</i>	Max current

46.3.6.19 **status_t SD_PollingCardStatusBusy (*sd_card_t * card*, *uint32_t timeoutMs*)**

This function can be used to polling the status from busy to Idle, the function will return if the card status idle or timeout.

Parameters

<i>card</i>	Card descriptor.
<i>timeoutMs</i>	polling card status timeout value.

Return values

<i>kStatus_Success</i>	Operate successfully.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	CMD13 transfer failed.
<i>kStatus_SDMMC_-PollingCardIdle-Failed,polling</i>	card DAT0 idle failed.

46.4 MMC Card Driver

46.4.1 Overview

The MMCCARD driver provide card initialization/read/write/erase interface.

46.4.2 MMC CARD Operation

error log support

Not support yet

User configurable

Board dependency

Mutual exclusive access support for RTOS

MMCCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization.

```
MMC_Deinit(card); /* This function will destroy the created mutex */
MMC_Init(card);
```

Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/sdmmc_examples/

Data Structures

- struct [mmc_usr_param_t](#)
card user parameter [More...](#)
- struct [mmc_card_t](#)
mmc card state [More...](#)

Macros

- #define [FSL_MMC_DRIVER_VERSION](#)([MAKE_VERSION](#)(2U, 5U, 0U)) /*2.5.0*/
Middleware mmc version.

Typedefs

- `typedef void(* mmc_io_strength_t)(uint32_t busFreq)`
card io strength control

Enumerations

- `enum {`
 `kMMC_SupportHighSpeed26MHZFlag = (1U << 0U),`
 `kMMC_SupportHighSpeed52MHZFlag = (1U << 1U),`
 `kMMC_SupportHighSpeedDDR52MHZ180V300VFlag = (1 << 2U),`
 `kMMC_SupportHighSpeedDDR52MHZ120VFlag = (1 << 3U),`
 `kMMC_SupportHS200200MHZ180VFlag = (1 << 4U),`
 `kMMC_SupportHS200200MHZ120VFlag = (1 << 5U),`
 `kMMC_SupportHS400DDR200MHZ180VFlag = (1 << 6U),`
 `kMMC_SupportHS400DDR200MHZ120VFlag = (1 << 7U),`
 `kMMC_SupportHighCapacityFlag = (1U << 8U),`
 `kMMC_SupportAlternateBootFlag = (1U << 9U),`
 `kMMC_SupportDDRBootFlag = (1U << 10U),`
 `kMMC_SupportHighSpeedBootFlag = (1U << 11U),`
 `kMMC_SupportEnhanceHS400StrobeFlag = (1U << 12U) }`
MMC card flags.
 `}`
 `kMMC_Sleep = 1U,`
 `kMMC_Awake = 0U }`
mmccard sleep/awake state

MMCCARD Function

- `status_t MMC_Init (mmc_card_t *card)`
Initializes the MMC card and host.
- `void MMC_Deinit (mmc_card_t *card)`
Deinitializes the card and host.
- `status_t MMC_CardInit (mmc_card_t *card)`
Initializes the card.
- `void MMC_CardDeinit (mmc_card_t *card)`
Deinitializes the card.
- `status_t MMC_HostInit (mmc_card_t *card)`
initialize the host.
- `void MMC_HostDeinit (mmc_card_t *card)`
Deinitializes the host.
- `void MMC_HostDoReset (mmc_card_t *card)`
Resets the host.
- `void MMC_HostReset (SDMMCHOST_CONFIG *host)`
Resets the host.
- `void MMC_SetCardPower (mmc_card_t *card, bool enable)`

- **Sets card power.**
• `bool MMC_CheckReadOnly (mmc_card_t *card)`
Checks if the card is read-only.
- `status_t MMC_ReadBlocks (mmc_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`
Reads data blocks from the card.
- `status_t MMC_WriteBlocks (mmc_card_t *card, const uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)`
Writes data blocks to the card.
- `status_t MMC_EraseGroups (mmc_card_t *card, uint32_t startGroup, uint32_t endGroup)`
Erases groups of the card.
- `status_t MMC_SelectPartition (mmc_card_t *card, mmc_access_partition_t partitionNumber)`
Selects the partition to access.
- `status_t MMC_SetBootConfig (mmc_card_t *card, const mmc_boot_config_t *config)`
Configures the boot activity of the card.
- `status_t MMC_StartBoot (mmc_card_t *card, const mmc_boot_config_t *mmcConfig, uint8_t *buffer, sdmmchost_boot_config_t *hostConfig)`
MMC card start boot.
- `status_t MMC_SetBootConfigWP (mmc_card_t *card, uint8_t wp)`
MMC card set boot configuration write protect.
- `status_t MMC_ReadBootData (mmc_card_t *card, uint8_t *buffer, sdmmchost_boot_config_t *hostConfig)`
MMC card continuous read boot data.
- `status_t MMC_StopBoot (mmc_card_t *card, uint32_t bootMode)`
MMC card stop boot mode.
- `status_t MMC_SetBootPartitionWP (mmc_card_t *card, mmc_boot_partition_wp_t bootPartitionWP)`
MMC card set boot partition write protect.
- `status_t MMC_EnableCacheControl (mmc_card_t *card, bool enable)`
MMC card cache control function.
- `status_t MMC_FlushCache (mmc_card_t *card)`
MMC card cache flush function.
- `status_t MMC_SetSleepAwake (mmc_card_t *card, mmc_sleep_awake_t state)`
MMC sets card sleep awake state.
- `status_t MMC_PollingCardStatusBusy (mmc_card_t *card, bool checkStatus, uint32_t timeoutMs)`
Polling card idle status.

46.4.3 Data Structure Documentation

46.4.3.1 struct mmc_usr_param_t

Data Fields

- `mmc_io_strength_t ioStrength`
switch sd io strength
- `uint32_t maxFreq`
board support maximum frequency
- `uint32_t capability`
board capability flag

46.4.3.2 struct mmc_card_t

Defines the card structure including the necessary fields to identify and describe the card.

Data Fields

- `sdmmchost_t * host`
Host information.
- `mmc_usr_param_t usrParam`
user parameter
- `bool isHostReady`
Use this flag to indicate if host re-init needed or not.
- `bool noInternalAlign`
Use this flag to disable sdmmc align.
- `uint32_t busClock_Hz`
MMC bus clock united in Hz.
- `uint32_t relativeAddress`
Relative address of the card.
- `bool enablePreDefinedBlockCount`
Enable PRE-DEFINED block count when read/write.
- `uint32_t flags`
Capability flag in `_mmc_card_flag`.
- `uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]`
raw buffer used for mmc driver internal
- `uint32_t ocr`
Raw OCR content.
- `mmc_cid_t cid`
CID.
- `mmc_csd_t csd`
CSD.
- `mmc_extended_csd_t extendedCsd`
Extended CSD.
- `uint32_t blockSize`
Card block size.
- `uint32_t userPartitionBlocks`
Card total block number in user partition.
- `uint32_t bootPartitionBlocks`
Boot partition size united as block size.
- `uint32_t eraseGroupBlocks`
Erase group size united as block size.
- `mmc_access_partition_t currentPartition`
Current access partition.
- `mmc_voltage_window_t hostVoltageWindowVCCQ`
application must set this value according to board specific
- `mmc_voltage_window_t hostVoltageWindowVCC`
application must set this value according to board specific
- `mmc_high_speed_timing_t busTiming`
indicates the current work timing mode
- `mmc_data_bus_width_t busWidth`
indicates the current work bus width

- `sdmmc_osa_mutex_t lock`
card access lock

Field Documentation

(1) `bool mmc_card_t::noInternalAlign`

If disabled, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are aligned to low level driver.

46.4.4 Macro Definition Documentation

46.4.4.1 `#define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 5U, 0U)) /*2.5.0*/`

46.4.5 Enumeration Type Documentation

46.4.5.1 anonymous enum

Enumerator

`kMMC_SupportHighSpeed26MHZFlag` Support high speed 26MHZ.
`kMMC_SupportHighSpeed52MHZFlag` Support high speed 52MHZ.
`kMMC_SupportHighSpeedDDR52MHZ180V300VFlag` ddr 52MHZ 1.8V or 3.0V
`kMMC_SupportHighSpeedDDR52MHZ120VFlag` DDR 52MHZ 1.2V.
`kMMC_SupportHS200200MHZ180VFlag` HS200 ,200MHZ,1.8V.
`kMMC_SupportHS200200MHZ120VFlag` HS200, 200MHZ, 1.2V.
`kMMC_SupportHS400DDR200MHZ180VFlag` HS400, DDR, 200MHZ,1.8V.
`kMMC_SupportHS400DDR200MHZ120VFlag` HS400, DDR, 200MHZ,1.2V.
`kMMC_SupportHighCapacityFlag` Support high capacity.
`kMMC_SupportAlternateBootFlag` Support alternate boot.
`kMMC_SupportDDRBootFlag` support DDR boot flag
`kMMC_SupportHighSpeedBootFlag` support high speed boot flag
`kMMC_SupportEnhanceHS400StrobeFlag` support enhance HS400 strobe

46.4.5.2 `enum mmc_sleep_awake_t`

Enumerator

`kMMC_Sleep` MMC card sleep.
`kMMC_Awake` MMC card awake.

46.4.6 Function Documentation

46.4.6.1 `status_t MMC_Init(mmc_card_t * card)`

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```
MMC_Deinit(card);
MMC_Init(card);
*
```

Return values

<i>kStatus_SDMMC_HostNotReady</i>	Host is not ready.
<i>kStatus_SDMMC_GoIdleFailed</i>	Going idle failed.
<i>kStatus_SDMMC_HandShakeOperationConditionFailed</i>	Sending operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Sending CID failed.
<i>kStatus_SDMMC_SetRelativeAddressFailed</i>	Setgng relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Sending CSD failed.
<i>kStatus_SDMMC_CardNotSupport</i>	Card not support.
<i>kStatus_SDMMC_SelectCardFailed</i>	Sending SELECT_CARD command failed.
<i>kStatus_SDMMC_SendExtendedCsdFailed</i>	Sending EXT_CSD failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Setting bus width failed.

<i>kStatus_SDMMC_Switch-BusTimingFailed</i>	Switching high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Setting card block size failed.
<i>kStatus_SDMMC_Set-PowerClassFail</i>	Setting card power class failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.2 void MMC_Deinit (mmc_card_t * *card*)

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.4.6.3 status_t MMC_CardInit (mmc_card_t * *card*)

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```
MMC_CardDeinit(card);
MMC_CardInit(card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	Host is not ready.
------------------------------------	--------------------

<i>kStatus_SDMMC_GoIdleFailed</i>	Going idle failed.
<i>kStatus_SDMMC_HandShakeOperationConditionFailed</i>	Sending operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Sending CID failed.
<i>kStatus_SDMMC_SetRelativeAddressFailed</i>	Setting relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Sending CSD failed.
<i>kStatus_SDMMC_CardNotSupport</i>	Card not support.
<i>kStatus_SDMMC_SelectCardFailed</i>	Sending SELECT_CARD command failed.
<i>kStatus_SDMMC_SendExtendedCsdFailed</i>	Sending EXT_CSD failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Setting bus width failed.
<i>kStatus_SDMMC_SwitchBusTimingFailed</i>	Switching high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Setting card block size failed.
<i>kStatus_SDMMC_SetPowerClassFail</i>	Setting card power class failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.4 void MMC_CardDeinit (mmc_card_t * **card**)

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.4.6.5 status_t MMC_HostInit (mmc_card_t * *card*)

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.4.6.6 void MMC_HostDeinit (mmc_card_t * *card*)

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.4.6.7 void MMC_HostDoReset (mmc_card_t * *card*)

This function resets the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.4.6.8 void MMC_HostReset (SDMMCHOST_CONFIG * *host*)

Deprecated Do not use this function. It has been superceded by [MMC_HostDoReset](#). This function resets the specific host.

Parameters

<i>host</i>	Host descriptor.
-------------	------------------

46.4.6.9 void MMC_SetCardPower (mmc_card_t * *card*, bool *enable*)

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	True is powering on, false is powering off.

46.4.6.10 bool MMC_CheckReadOnly (mmc_card_t * *card*)

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>true</i>	Card is read only.
<i>false</i>	Card isn't read only.

46.4.6.11 status_t MMC_ReadBlocks (mmc_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save data.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to read.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.

<i>kStatus_SDMMC_SetBlockCountFailed</i>	Setting block count failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_StopTransmissionFailed</i>	Stopping transmission failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.12 **status_t MMC_WriteBlocks (mmc_card_t * *card*, const uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)**

Note

1. It is a thread safe function.
2. It is an async write function which means that the card status may still be busy after the function returns. Application can call function MMC_PollingCardStatusBusy to wait for the card status to be idle after the write operation.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save data blocks.
<i>startBlock</i>	Start block number to write.
<i>blockCount</i>	Block count.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_NotSupportYet</i>	Not support now.
<i>kStatus_SDMMC_SetBlockCountFailed</i>	Setting block count failed.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Sending status failed.

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_StopTransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.13 status_t MMC_EraseGroups (*mmc_card_t * card*, *uint32_t startGroup*, *uint32_t endGroup*)

The erase command is best used to erase the entire device or a partition. Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

Note

1. It is a thread safe function.
2. This function always polls card busy status according to the timeout value defined in the card register after all the erase command sent out.

Parameters

<i>card</i>	Card descriptor.
<i>startGroup</i>	Start group number.
<i>endGroup</i>	End group number.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.14 status_t MMC_SelectPartition (*mmc_card_t * card*, *mmc_access_partition_t partitionNumber*)

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>partition-Number</i>	The partition number.

Return values

<i>kStatus_SDMMC_ConfigureExtendedCsdFailed</i>	Configuring EXT_CSD failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.15 status_t MMC_SetBootConfig (mmc_card_t * *card*, const mmc_boot_config_t * *config*)

Parameters

<i>card</i>	Card descriptor.
<i>config</i>	Boot configuration structure.

Return values

<i>kStatus_SDMMC_NotSupportYet</i>	Not support now.
<i>kStatus_SDMMC_ConfigureExtendedCsdFailed</i>	Configuring EXT_CSD failed.
<i>kStatus_SDMMC_ConfigureBootFailed</i>	Configuring boot failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.16 status_t MMC_StartBoot (mmc_card_t * *card*, const mmc_boot_config_t * *mmcConfig*, uint8_t * *buffer*, sdmmchost_boot_config_t * *hostConfig*)

Parameters

<i>card</i>	Card descriptor.
<i>mmcConfig</i>	The mmc Boot configuration structure.
<i>buffer</i>	Address to receive data.
<i>hostConfig</i>	Host boot configurations.

Return values

<i>kStatus_Fail</i>	Failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Go_IdleFailed</i>	Resetting card failed.
<i>kStatus_Success</i>	Operation succeeded.

46.4.6.17 status_t MMC_SetBootConfigWP (*mmc_card_t * card, uint8_t wp*)

Parameters

<i>card</i>	Card descriptor.
<i>wp</i>	Write protect value.

46.4.6.18 status_t MMC_ReadBootData (*mmc_card_t * card, uint8_t * buffer, sdmmchost_boot_config_t * hostConfig*)

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	Buffer address.
<i>hostConfig</i>	Host boot configurations.

46.4.6.19 status_t MMC_StopBoot (*mmc_card_t * card, uint32_t bootMode*)

Parameters

<i>card</i>	Card descriptor.
<i>bootMode</i>	Boot mode.

46.4.6.20 status_t MMC_SetBootPartitionWP (*mmc_card_t * card*, *mmc_boot_partition_wp_t bootPartitionWP*)

Parameters

<i>card</i>	Card descriptor.
<i>bootPartition-WP</i>	Boot partition write protect value.

46.4.6.21 status_t MMC_EnableCacheControl (*mmc_card_t * card*, *bool enable*)

The mmc device's cache is enabled by the driver by default. The cache should in typical case reduce the access time (compared to an access to the main nonvolatile storage) for both write and read.

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	True is enabling the cache, false is disabling the cache.

46.4.6.22 status_t MMC_FlushCache (*mmc_card_t * card*)

A Flush operation refers to the requirement, from the host to the device, to write the cached data to the nonvolatile memory. Prior to a flush, the device may autonomously write data to the nonvolatile memory, but after the flush operation all data in the volatile area must be written to nonvolatile memory. There is no requirement for flush due to switching between the partitions. (Note: This also implies that the cache data shall not be lost when switching between partitions). Cached data may be lost in SLEEP state, so host should flush the cache before placing the device into SLEEP state.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

46.4.6.23 status_t MMC_SetSleepAwake (mmc_card_t * card, mmc_sleep_awake_t state)

The Sleep/Awake command is used to initiate the state transition between Standby state and Sleep state. The memory device indicates the transition phase busy by pulling down the DAT0 line. The Sleep/Standby state is reached when the memory device stops pulling down the DAT0 line, then the function returns.

Parameters

<i>card</i>	Card descriptor.
<i>state</i>	The sleep/awake command argument, refer to mmc_sleep_awake_t .

Return values

<i>kStatus_SDMMC_NotSupportYet</i>	Indicates the memory device doesn't support the Sleep/Awake command.
<i>kStatus_SDMMC_TransferFailed</i>	Indicates command transferred fail.
<i>kStatus_SDMMC_PollingCardIdleFailed</i>	Indicates polling DAT0 busy timeout.
<i>kStatus_SDMMC_DeselectCardFailed</i>	Indicates deselect card command failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Indicates select card command failed.
<i>kStatus_Success</i>	Indicates the card state switched successfully.

46.4.6.24 status_t MMC_PollingCardStatusBusy (mmc_card_t * card, bool checkStatus, uint32_t timeoutMs)

This function can be used to poll the status from busy to idle, the function will return with the card status being idle or timeout or command failed.

Parameters

<i>card</i>	Card descriptor.
<i>checkStatus</i>	True is send CMD and read DAT0 status to check card status, false is read DAT0 status only.
<i>timeoutMs</i>	Polling card status timeout value.

Return values

<i>kStatus_SDMMC_Card-StatusIdle</i>	Card is idle.
<i>kStatus_SDMMC_Card-StatusBusy</i>	Card is busy.
<i>kStatus_SDMMC_TransferFailed</i>	Command transfer failed.
<i>kStatus_SDMMC_SwitchFailed</i>	Status command reports switch error.

46.5 SDMMC HOST Driver

46.5.1 Overview

The host adapter driver provide adapter for blocking/non_blocking mode.

Modules

- [USDHC HOST adapter Driver](#)

46.6 SDMMC OSA

46.6.1 Overview

The sdmmc osa adapter provide interface of os adapter.

Data Structures

- struct `sdmmc_osa_event_t`
sdmmc osa event [More...](#)
- struct `sdmmc_osa_mutex_t`
sdmmc osa mutex [More...](#)

Macros

- #define `SDMMC_OSA_EVENT_TRANSFER_CMD_SUCCESS` (1UL << 0U)
transfer event
- #define `SDMMC_OSA_EVENT_CARD_INSERTED` (1UL << 8U)
card detect event, start from index 8
- #define `SDMMC_OSA_POLLING_EVENT_BY_SEMPHORE` 1
enable semaphore by default

sdmmc osa Function

- void `SDMMC_OSAInit` (void)
Initialize OSA.
- `status_t SDMMC_OSAEventCreate` (void *eventHandle)
OSA Create event.
- `status_t SDMMC_OSAEventWait` (void *eventHandle, uint32_t eventType, uint32_t timeoutMilliseconds, uint32_t *event)
Wait event.
- `status_t SDMMC_OSAEventSet` (void *eventHandle, uint32_t eventType)
set event.
- `status_t SDMMC_OSAEventGet` (void *eventHandle, uint32_t eventType, uint32_t *flag)
Get event flag.
- `status_t SDMMC_OSAEventClear` (void *eventHandle, uint32_t eventType)
clear event flag.
- `status_t SDMMC_OSAEventDestroy` (void *eventHandle)
Delete event.
- `status_t SDMMC_OSAMutexCreate` (void *mutexHandle)
Create a mutex.
- `status_t SDMMC_OSAMutexLock` (void *mutexHandle, uint32_t millisec)
set event.
- `status_t SDMMC_OSAMutexUnlock` (void *mutexHandle)
Get event flag.
- `status_t SDMMC_OSAMutexDestroy` (void *mutexHandle)
Delete mutex.

- void **SDMMC_OSA****Delay** (uint32_t milliseconds)
sdmmc delay.
- uint32_t **SDMMC_OSA****DelayUs** (uint32_t microseconds)
sdmmc delay us.

46.6.2 Data Structure Documentation

46.6.2.1 struct sdmmc_osa_event_t

46.6.2.2 struct sdmmc_osa_mutex_t

46.6.3 Function Documentation

46.6.3.1 status_t **SDMMC_OSAEventCreate** (void * *eventHandle*)

Parameters

<i>eventHandle</i>	event handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.2 status_t **SDMMC_OSAEventWait** (void * *eventHandle*, uint32_t *eventType*, uint32_t *timeoutMilliseconds*, uint32_t * *event*)

Parameters

<i>eventHandle</i>	The event type
<i>eventType</i>	Timeout time in milliseconds.
<i>timeout-Milliseconds</i>	timeout value in ms.
<i>event</i>	event flags.

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.3 status_t SDMMC_OSAEventSet (void * *eventHandle*, uint32_t *eventType*)

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	The event type

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.4 status_t SDMMC_OSAEventGet (void * *eventHandle*, uint32_t *eventType*, uint32_t * *flag*)

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	event type.
<i>flag</i>	pointer to store event value.

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.5 status_t SDMMC_OSAEventClear (void * *eventHandle*, uint32_t *eventType*)

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	The event type

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.6 status_t SDMMC_OSAEventDestroy (void * *eventHandle*)

Parameters

<i>eventHandle</i>	The event handle.
--------------------	-------------------

46.6.3.7 status_t SDMMC_OSAMutexCreate (void * *mutexHandle*)

Parameters

<i>mutexHandle</i>	mutex handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.8 status_t SDMMC_OSAMutexLock (void * *mutexHandle*, uint32_t *millisec*)

Parameters

<i>mutexHandle</i>	mutex handle.
<i>millisec</i>	The maximum number of milliseconds to wait for the mutex. If the mutex is locked, Pass the value <i>osaWaitForever_c</i> will wait indefinitely, pass 0 will return <i>KOSA_StatusTimeout</i> immediately.

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.9 status_t SDMMC_OSAMutexUnlock (void * *mutexHandle*)

Parameters

<i>mutexHandle</i>	mutex handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

46.6.3.10 **status_t SDMMC_OSAMutexDestroy(void * *mutexHandle*)**

Parameters

<i>mutexHandle</i>	The mutex handle.
--------------------	-------------------

46.6.3.11 **void SDMMC_OSADelay(uint32_t *milliseconds*)**

Parameters

<i>milliseconds</i>	time to delay
---------------------	---------------

46.6.3.12 **uint32_t SDMMC_OSADelayUs(uint32_t *microseconds*)**

Parameters

<i>microseconds</i>	time to delay
---------------------	---------------

Returns

actual delayed microseconds

46.6.4 USDHC HOST adapter Driver

46.6.4.1 Overview

The USDHC host adapter driver provide adapter for blocking/non_blocking mode.

Cache maintain capability

To maintain data integrity during DMA operations on the platform that has cache, host driver provide a cache maintain functionality by set: host.enableCacheControl = kSDMMCHOST_CacheControlRW-Buffer This is used for only when the low level driver cache maintain functionality is disabled. It is suggest that the address of buffer used for read/write is align with cache line size.

Cache line alignment maintain capability

when application submit a transfer request that the data buffer address is not align with cache line size, the potential data loss may happen during driver maintain the data cache, to avoid such issue happens, sdmmc usdhc host driver provides support on convert the unalign data transfer into align data transfer, if application would like to use the feature, please enable this functionality by define below macro firstly, #define SDMMCHOST_ENABLE_CACHE_LINE_ALIGN_TRANSFER 1 And then call SDMMCHOST_InstallCacheAlignBuffer to install a cache line size align buffer, please note the installed buffer size must not small than 2 * cache line size. Please note that this functionality is support by the non blocking adapter only.

Data Structures

- struct `sdmmchost_t`
sdmmc host handler [More...](#)

Macros

- #define `FSL_SDMMC_HOST_ADAPTER_VERSION` (`MAKE_VERSION(2U, 6U, 3U)`) /*2.6.-3*/
Middleware adapter version.
- #define `SDMMCHOST_SUPPORT_HIGH_SPEED` (1U)
sdmmc host misc capability
- #define `SDMMCHOST_SUPPORT_DDR50` (`SDMMCHOST_SUPPORT_DDR_MODE`)
sdmmc host sdcard DDR50 mode capability
- #define `SDMMCHOST_SUPPORT_SDR104` (1U)
sdmmc host sdcard SDR50 mode capability
- #define `SDMMCHOST_SUPPORT_SDR50` (1U)
sdmmc host sdcard SDR104/mmccard HS200 mode capability
- #define `SDMMCHOST_SUPPORT_HS400` (1U)
sdmmc host mmccard HS400 mode capability

- #define **SDMMCHOST_INSTANCE_SUPPORT_8_BIT_WIDTH**(host) FSL_FEATURE_USDH-C_INSTANCE_SUPPORT_8_BIT_WIDTHn(host->hostController.base)
sdmmc host instance capability
- #define **SDMMCHOST_DATA3_DETECT_CARD_DELAY** (10U)
sdmmchost delay for DAT3 detect card
- #define **SDMMCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE** (4U)
SDMMC host dma descriptor buffer address align size.
- #define **SDMMCHOST_STANDARD_TUNING_START** (10U)
tuning configuration
- #define **SDMMCHOST_TUINIG_STEP** (2U)
standard tuning stBep

Typedefs

- typedef **usdhc_transfer_t** **sdmmchost_transfer_t**
sdmmc host transfer function

Enumerations

- enum {
 kSDMMCHOST_SupportHighSpeed = 1U << 0U,
 kSDMMCHOST_SupportSuspendResume = 1U << 1U,
 kSDMMCHOST_SupportVoltage3v3 = 1U << 2U,
 kSDMMCHOST_SupportVoltage3v0 = 1U << 3U,
 kSDMMCHOST_SupportVoltage1v8 = 1U << 4U,
 kSDMMCHOST_SupportVoltage1v2 = 1U << 5U,
 kSDMMCHOST_Support4BitDataWidth = 1U << 6U,
 kSDMMCHOST_Support8BitDataWidth = 1U << 7U,
 kSDMMCHOST_SupportDDRMode = 1U << 8U,
 kSDMMCHOST_SupportDetectCardByData3 = 1U << 9U,
 kSDMMCHOST_SupportDetectCardByCD = 1U << 10U,
 kSDMMCHOST_SupportAutoCmd12 = 1U << 11U,
 kSDMMCHOST_SupportSDR104 = 1U << 12U,
 kSDMMCHOST_SupportSDR50 = 1U << 13U,
 kSDMMCHOST_SupportHS200 = 1U << 14U,
 kSDMMCHOST_SupportHS400 = 1U << 15U }

sdmmc host capability
- enum {
 kSDMMCHOST_EndianModeBig = 0U,
 kSDMMCHOST_EndianModeHalfWordBig = 1U,
 kSDMMCHOST_EndianModeLittle = 2U }

host Endian mode corresponding to driver define
- enum {
 kSDMMCHOST_StandardTuning = 0U,
 kSDMMCHOST_ManualTuning = 1U }

sdmmc host tuning type

- enum {

kSDMMCHOST_NoCacheControl = 0U,
 kSDMMCHOST_CacheControlRWBuffer = 1U }

sdmmc host maintain cache flag

USDHC host controller function

- void **SDMMCHOST_SetCardBusWidth** (sdmmhost_t *host, uint32_t dataBusWidth)

set data bus width.
- static void **SDMMCHOST_SendCardActive** (sdmmhost_t *host)

Send initialization active 80 clocks to card.
- static uint32_t **SDMMCHOST_SetCardClock** (sdmmhost_t *host, uint32_t targetClock)

Set card bus clock.
- static bool **SDMMCHOST_IsCardBusy** (sdmmhost_t *host)

check card status by DATA0.
- static uint32_t **SDMMCHOST_GetSignalLineStatus** (sdmmhost_t *host, uint32_t signalLine)

Get signal line status.
- static void **SDMMCHOST_EnableCardInt** (sdmmhost_t *host, bool enable)

enable card interrupt.
- static void **SDMMCHOST_EnableDDRMode** (sdmmhost_t *host, bool enable, uint32_t nibblePos)

enable DDR mode.
- static void **SDMMCHOST_EnableHS400Mode** (sdmmhost_t *host, bool enable)

enable HS400 mode.
- static void **SDMMCHOST_EnableStrobeDll** (sdmmhost_t *host, bool enable)

enable STROBE DLL.
- status_t **SDMMCHOST_StartBoot** (sdmmhost_t *host, sdmmhost_boot_config_t *hostConfig, sdmmhost_cmd_t *cmd, uint8_t *buffer)

start read boot data.
- status_t **SDMMCHOST_ReadBootData** (sdmmhost_t *host, sdmmhost_boot_config_t *hostConfig, uint8_t *buffer)

read boot data.
- static void **SDMMCHOST_EnableBoot** (sdmmhost_t *host, bool enable)

enable boot mode.
- status_t **SDMMCHOST_CardIntInit** (sdmmhost_t *host, void *sdioInt)

card interrupt function.
- static void **SDMMCHOST_ForceClockOn** (sdmmhost_t *host, bool enable)

force card clock on.
- void **SDMMCHOST_SwitchToVoltage** (sdmmhost_t *host, uint32_t voltage)

switch to voltage.
- status_t **SDMMCHOST_CardDetectInit** (sdmmhost_t *host, void *cd)

card detect init function.
- status_t **SDMMCHOST_PollingCardDetectStatus** (sdmmhost_t *host, uint32_t waitCardStatus, uint32_t timeout)

Detect card insert, only need for SD cases.
- uint32_t **SDMMCHOST_CardDetectStatus** (sdmmhost_t *host)

card detect status.
- status_t **SDMMCHOST_Init** (sdmmhost_t *host)

Init host controller.
- void **SDMMCHOST_Deinit** (sdmmhost_t *host)

- *Deinit host controller.*
- void **SDMMCHOST_SetCardPower** (**sdmmchost_t** *host, bool enable)
host power off card function.
- **status_t SDMMCHOST_TransferFunction** (**sdmmchost_t** *host, **sdmmchost_transfer_t** *content)
host transfer function.
- **status_t SDMMCHOST_ExecuteTuning** (**sdmmchost_t** *host, uint32_t tuningCmd, uint32_t *revBuf, uint32_t blockSize)
sdmmc host execute tuning.
- void **SDMMCHOST_Reset** (**sdmmchost_t** *host)
host reset function.
- void **SDMMCHOST_ConvertDataToLittleEndian** (**sdmmchost_t** *host, uint32_t *data, uint32_t wordSize, uint32_t format)
sdmmc host convert data sequence to little endian sequence

46.6.4.2 Data Structure Documentation

46.6.4.2.1 struct **sdmmchost_t**

Data Fields

- **usdhc_host_t hostController**
host configuration
- void * **dmaDesBuffer**
DMA descriptor buffer address.
- uint32_t **dmaDesBufferWordsNum**
DMA descriptor buffer size in byte.
- usdhc_handle_t **handle**
host controller handler
- uint32_t **capability**
host controller capability
- uint32_t **maxBlockCount**
host controller maximum block count
- uint32_t **maxBlockSize**
host controller maximum block size
- uint8_t **tuningType**
host tuning type
- **sdmmc_osa_event_t hostEvent**
host event handler
- void * **cd**
card detect
- void * **cardInt**
call back function for card interrupt
- **sdmmc_osa_mutex_t lock**
host access lock

46.6.4.3 Macro Definition Documentation

**46.6.4.3.1 #define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 6U, 3U))
/*2.6.3*/**

46.6.4.3.2 #define SDMMCHOST_STANDARD_TUNING_START (10U)

standard tuning start point

46.6.4.4 Enumeration Type Documentation

46.6.4.4.1 anonymous enum

Enumerator

kSDMMCHOST_SupportHighSpeed high speed capability
kSDMMCHOST_SupportSuspendResume suspend resume capability
kSDMMCHOST_SupportVoltage3v3 3V3 capability
kSDMMCHOST_SupportVoltage3v0 3V0 capability
kSDMMCHOST_SupportVoltage1v8 1V8 capability
kSDMMCHOST_SupportVoltage1v2 1V2 capability
kSDMMCHOST_Support4BitDataWidth 4 bit data width capability
kSDMMCHOST_Support8BitDataWidth 8 bit data width capability
kSDMMCHOST_SupportDDRMode DDR mode capability.
kSDMMCHOST_SupportDetectCardByData3 data3 detect card capability
kSDMMCHOST_SupportDetectCardByCD CD detect card capability.
kSDMMCHOST_SupportAutoCmd12 auto command 12 capability
kSDMMCHOST_SupportSDR104 SDR104 capability.
kSDMMCHOST_SupportSDR50 SDR50 capability.
kSDMMCHOST_SupportHS200 HS200 capability.
kSDMMCHOST_SupportHS400 HS400 capability.

46.6.4.4.2 anonymous enum

Enumerator

kSDMMCHOST_EndianModeBig Big endian mode.
kSDMMCHOST_EndianModeHalfWordBig Half word big endian mode.
kSDMMCHOST_EndianModeLittle Little endian mode.

46.6.4.4.3 anonymous enum

Enumerator

kSDMMCHOST_StandardTuning standard tuning type
kSDMMCHOST_ManualTuning manual tuning type

46.6.4.4 anonymous enum

Enumerator

kSDMMCHOST_NoCacheControl sdmmc host cache control disabled

kSDMMCHOST_CacheControlRWBuffer sdmmc host cache control read/write buffer

46.6.4.5 Function Documentation

46.6.4.5.1 void SDMMCHOST_SetCardBusWidth (*sdmmchost_t * host*, *uint32_t dataBusWidth*)

Parameters

<i>host</i>	host handler
<i>dataBusWidth</i>	data bus width

46.6.4.5.2 static void SDMMCHOST_SendCardActive (*sdmmchost_t * host*) [inline], [static]

Parameters

<i>host</i>	host handler
-------------	--------------

46.6.4.5.3 static uint32_t SDMMCHOST_SetCardClock (*sdmmchost_t * host*, *uint32_t targetClock*) [inline], [static]

Parameters

<i>host</i>	host handler
<i>targetClock</i>	target clock frequency

Return values

<i>actual</i>	clock frequency can be reach.
---------------	-------------------------------

46.6.4.5.4 static bool SDMMCHOST_IsCardBusy (*sdmmchost_t * host*) [inline], [static]

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>true</i>	is busy, false is idle.
-------------	-------------------------

46.6.4.5.5 static uint32_t SDMMCHOST_GetSignalLineStatus (*sdmmchost_t * host, uint32_t signalLine*) [inline], [static]

Parameters

<i>host</i>	host handler
<i>signalLine</i>	signal line type, reference _sdmmc_signal_line

46.6.4.5.6 static void SDMMCHOST_EnableCardInt (*sdmmchost_t * host, bool enable*) [inline], [static]

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

46.6.4.5.7 static void SDMMCHOST_EnableDDRMode (*sdmmchost_t * host, bool enable, uint32_t nibblePos*) [inline], [static]

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.
<i>nibblePos</i>	nibble position indication. 0- the sequence is 'odd high nibble -> even high nibble -> odd low nibble -> even low nibble'; 1- the sequence is 'odd high nibble -> odd low nibble -> even high nibble -> even low nibble'.

46.6.4.5.8 static void SDMMCHOST_EnableHS400Mode (*sdmmchost_t * host, bool enable*) [inline], [static]

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

**46.6.4.5.9 static void SDMMCHOST_EnableStrobeDII (*sdmmchost_t * host, bool enable*)
[inline], [static]**

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

46.6.4.5.10 status_t SDMMCHOST_StartBoot (*sdmmchost_t * host, sdmmchost_boot_config_t * hostConfig, sdmmchost_cmd_t * cmd, uint8_t * buffer*)

Parameters

<i>host</i>	host handler
<i>hostConfig</i>	boot configuration
<i>cmd</i>	boot command
<i>buffer</i>	buffer address

46.6.4.5.11 status_t SDMMCHOST_ReadBootData (*sdmmchost_t * host, sdmmchost_boot_config_t * hostConfig, uint8_t * buffer*)

Parameters

<i>host</i>	host handler
<i>hostConfig</i>	boot configuration
<i>buffer</i>	buffer address

**46.6.4.5.12 static void SDMMCHOST_EnableBoot (*sdmmchost_t * host, bool enable*)
[inline], [static]**

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable

46.6.4.5.13 status_t SDMMCHOST_CardIntInit (*sdmmchost_t * host, void * sdioInt*)

Parameters

<i>host</i>	host handler
<i>sdioInt</i>	card interrupt configuration

46.6.4.5.14 static void SDMMCHOST_ForceClockOn (*sdmmchost_t * host, bool enable*) [inline], [static]

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

46.6.4.5.15 void SDMMCHOST_SwitchToVoltage (*sdmmchost_t * host, uint32_t voltage*)

Parameters

<i>host</i>	host handler
<i>voltage</i>	switch to voltage level.

46.6.4.5.16 status_t SDMMCHOST_CardDetectInit (*sdmmchost_t * host, void * cd*)

Parameters

<i>host</i>	host handler
-------------	--------------

<i>cd</i>	card detect configuration
-----------	---------------------------

46.6.4.5.17 status_t SDMMCHOST_PollingCardDetectStatus (*sdmmchost_t * host, uint32_t waitCardStatus, uint32_t timeout*)

Parameters

<i>host</i>	host handler
<i>waitCardStatus</i>	status which user want to wait
<i>timeout</i>	wait time out.

Return values

<i>kStatus_Success</i>	detect card insert
<i>kStatus_Fail</i>	card insert event fail

46.6.4.5.18 uint32_t SDMMCHOST_CardDetectStatus (*sdmmchost_t * host*)

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kSD_Inserted,kSD_Removed</i>	
---------------------------------	--

46.6.4.5.19 status_t SDMMCHOST_Init (*sdmmchost_t * host*)

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDMMCHOST_Deinit(host);
* SDMMCHOST_Init(host);
*
```

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kStatus_Success</i>	host init success
<i>kStatus_Fail</i>	event fail

46.6.4.5.20 void SDMMCHOST_Deinit (*sdmmchost_t * host*)

Please note it is a thread safe function.

Parameters

<i>host</i>	host handler
-------------	--------------

46.6.4.5.21 void SDMMCHOST_SetCardPower (*sdmmchost_t * host, bool enable*)

Parameters

<i>host</i>	host handler
<i>enable</i>	true is power on, false is power down.

46.6.4.5.22 *status_t SDMMCHOST_TransferFunction (*sdmmchost_t * host, sdmmchost_transfer_t * content*)*

Please note it is a thread safe function.

Note

the host transfer function support below functionality,

1. Non-cache line size alignment check on the data buffer, it is means that no matter the data buffer used for data transfer is align with cache line size or not, sdmmc host driver will use the address directly.
2. Cache line size alignment check on the data buffer, sdmmc host driver will check the data buffer address, if the buffer is not align with cache line size, sdmmc host driver will convert it to cache line size align buffer, the functionality is enabled by #define SDMMCHOST_ENABLE_CACHE_LINE_ALIGN_TRANSFER 1 #define FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER 1U If application would like to enable the cache line size align functionality, please make sure the SDMMCHOST_InstallCacheAlignBuffer is called before

submit data transfer request and make sure the installing buffer size is not smaller than $2 * \text{cache line size}$.

Parameters

<i>host</i>	host handler
<i>content</i>	transfer content.

46.6.4.5.23 **status_t SDMMCHOST_ExecuteTuning (*sdmmchost_t * host, uint32_t tuningCmd, uint32_t * revBuf, uint32_t blockSize*)**

Parameters

<i>host</i>	host handler
<i>tuningCmd</i>	tuning command.
<i>revBuf</i>	receive buffer pointer
<i>blockSize</i>	tuning data block size.

46.6.4.5.24 **void SDMMCHOST_Reset (*sdmmchost_t * host*)**

Parameters

<i>host</i>	host handler
-------------	--------------

46.6.4.5.25 **void SDMMCHOST_ConvertDataToLittleEndian (*sdmmchost_t * host, uint32_t * data, uint32_t wordSize, uint32_t format*)**

Parameters

<i>host</i>	host handler.
<i>data</i>	data buffer address.
<i>wordSize</i>	data buffer size in word.
<i>format</i>	data packet format.

46.7 SDMMC Common

46.7.1 Overview

The sdmmc common function and definition.

Data Structures

- struct `sd_detect_card_t`
`sd card detect` [More...](#)
- struct `sd_io_voltage_t`
`io voltage control configuration` [More...](#)
- struct `sd_usr_param_t`
`sdcard user parameter` [More...](#)
- struct `sdio_card_int_t`
`card interrupt application callback` [More...](#)
- struct `sdio_usr_param_t`
`sdio user parameter` [More...](#)
- struct `sdio_fbr_t`
`sdio card FBR register` [More...](#)
- struct `sdio_common_cis_t`
`sdio card common CIS` [More...](#)
- struct `sdio_func_cis_t`
`sdio card function CIS` [More...](#)
- struct `sd_status_t`
`SD card status.` [More...](#)
- struct `sd_cid_t`
`SD card CID register.` [More...](#)
- struct `sd_csd_t`
`SD card CSD register.` [More...](#)
- struct `sd_scr_t`
`SD card SCR register.` [More...](#)
- struct `mmc_cid_t`
`MMC card CID register.` [More...](#)
- struct `mmc_csd_t`
`MMC card CSD register.` [More...](#)
- struct `mmc_extended_csd_t`
`MMC card Extended CSD register (unit: byte).` [More...](#)
- struct `mmc_extended_csd_config_t`
`MMC Extended CSD configuration.` [More...](#)
- struct `mmc_boot_config_t`
`MMC card boot configuration definition.` [More...](#)

Macros

- #define `SWAP_WORD_BYTSEQUENCE(x)` (`__REV(x)`)
`Reverse byte sequence in uint32_t.`
- #define `SWAP_HALF_WROD_BYTSEQUENCE(x)` (`__REV16(x)`)

- Reverse byte sequence for each half word in `uint32_t`.
- `#define FSL_SDMMC_MAX_VOLTAGE_RETRIES` (1000U)
Maximum loop count to check the card operation voltage range.
- `#define FSL_SDMMC_MAX_CMD_RETRIES` (10U)
Maximum loop count to send the cmd.
- `#define FSL_SDMMC_DEFAULT_BLOCK_SIZE` (512U)
Default block size.
- `#define SDMMC_DATA_BUFFER_ALIGN_CACHE` `FSL_FEATURE_L1DCACHE_LINESIZE_BYT`
make sure the internal buffer address is cache align
- `#define FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE` (`FSL_SDMMC_DEFAULT_BLOCK_SIZE + SDMMC_DATA_BUFFER_ALIGN_CACHE`)
sdmmc card internal buffer size
- `#define FSL_SDMMC_CARD_MAX_BUS_FREQ`(max, target) ((max) == 0U ? (target) : ((max) > (target) ? (target) : (max)))
get maximum freq
- `#define SDMMC_LOG`(format,...)
SD/MMC error log.
- `#define SDMMC_CLOCK_400KHZ` (4000000U)
SD/MMC card initialization clock frequency.
- `#define SD_CLOCK_25MHZ` (25000000U)
SD card bus frequency 1 in high-speed mode.
- `#define SD_CLOCK_50MHZ` (50000000U)
SD card bus frequency 2 in high-speed mode.
- `#define SD_CLOCK_100MHZ` (100000000U)
SD card bus frequency in SDR50 mode.
- `#define SD_CLOCK_208MHZ` (208000000U)
SD card bus frequency in SDR104 mode.
- `#define MMC_CLOCK_26MHZ` (26000000U)
MMC card bus frequency 1 in high-speed mode.
- `#define MMC_CLOCK_52MHZ` (52000000U)
MMC card bus frequency 2 in high-speed mode.
- `#define MMC_CLOCK_DDR52` (52000000U)
MMC card bus frequency in high-speed DDR52 mode.
- `#define MMC_CLOCK_HS200` (200000000U)
MMC card bus frequency in high-speed HS200 mode.
- `#define MMC_CLOCK_HS400` (400000000U)
MMC card bus frequency in high-speed HS400 mode.
- `#define SDMMC_MASK`(bit) (1UL << (bit))
mask convert
- `#define SDMMC_R1_ALL_ERROR_FLAG`
R1 all the error flag.
- `#define SDMMC_R1_CURRENT_STATE`(x) (((x)&0x00001E00U) >> 9U)
R1: current state.
- `#define SDSPI_R7_VERSION_SHIFT` (28U)
The bit mask for COMMAND VERSION field in R7.
- `#define SDSPI_R7_VERSION_MASK` (0xFU)
The bit mask for COMMAND VERSION field in R7.
- `#define SDSPI_R7_VOLTAGE_SHIFT` (8U)
The bit shift for VOLTAGE ACCEPTED field in R7.
- `#define SDSPI_R7_VOLTAGE_MASK` (0xFU)

- `#define SDSPI_R7_VOLTAGE_27_36_MASK (0x1U << SDSPI_R7_VOLTAGE_SHIFT)`
The bit mask for VOLTAGE ACCEPTED field in R7.
- `#define SDSPI_R7_VOLTAGE_2.7V_to_3.6V_MASK (0x1U << SDSPI_R7_VOLTAGE_SHIFT)`
The bit mask for VOLTAGE 2.7V to 3.6V field in R7.
- `#define SDSPI_R7_ECHO_SHIFT (0U)`
The bit shift for ECHO field in R7.
- `#define SDSPI_R7_ECHO_MASK (0xFFU)`
The bit mask for ECHO field in R7.
- `#define SDSPI_DATA_ERROR_TOKEN_MASK (0xFU)`
Data error token mask.
- `#define SDSPI_DATA_RESPONSE_TOKEN_MASK (0x1FU)`
Mask for data response bits.
- `#define SDIO_CCCR_REG_NUMBER (0x16U)`
sdio card cccr register number
- `#define SDIO_IO_READY_TIMEOUT_UNIT (10U)`
sdio IO ready timeout steps
- `#define SDIO_CMD_ARGUMENT_RW_POS (31U)`
read/write flag position
- `#define SDIO_CMD_ARGUMENT_FUNC_NUM_POS (28U)`
function number position
- `#define SDIO_DIRECT_CMD_ARGUMENT_RAW_POS (27U)`
direct raw flag position
- `#define SDIO_CMD_ARGUMENT_REG_ADDR_POS (9U)`
direct reg addr position
- `#define SDIO_CMD_ARGUMENT_REG_ADDR_MASK (0x1FFFFU)`
direct reg addr mask
- `#define SDIO_DIRECT_CMD_DATA_MASK (0xFFU)`
data mask
- `#define SDIO_EXTEND_CMD_ARGUMENT_BLOCK_MODE_POS (27U)`
extended command argument block mode bit position
- `#define SDIO_EXTEND_CMD_ARGUMENT_OP_CODE_POS (26U)`
extended command argument OP Code bit position
- `#define SDIO_EXTEND_CMD_BLOCK_MODE_MASK (0x08000000U)`
block mode mask
- `#define SDIO_EXTEND_CMD_OP_CODE_MASK (0x04000000U)`
op code mask
- `#define SDIO_EXTEND_CMD_COUNT_MASK (0x1FFU)`
byte/block count mask
- `#define SDIO_MAX_BLOCK_SIZE (2048U)`
max block size
- `#define SDIO_FBR_BASE(x) ((x)*0x100U)`
function basic register
- `#define SDIO_TPL_CODE_END (0xFFU)`
tuple end
- `#define SDIO_TPL_CODE_MANIFID (0x20U)`
manufacturer ID
- `#define SDIO_TPL_CODE_FUNCID (0x21U)`
function ID
- `#define SDIO_TPL_CODE_FUNCE (0x22U)`
function extension tuple
- `#define SDIO_OCR_VOLTAGE_WINDOW_MASK (0xFFFFU << 8U)`
sdio ocr voltage window mask

- #define **SDIO_OCR_IO_NUM_MASK** (7U << kSDIO_OcrIONumber)

sdio ocr register IO NUMBER mask
- #define **SDIO_CCCR_SUPPORT_HIGHSPEED** (1UL << 9U)

UHS timing mode flag.
- #define **SDIO_CCCR_DRIVER_TYPE_MASK** (3U << 4U)

Driver type flag.
- #define **SDIO_CCCR_ASYNC_INT_MASK** (1U)

async interrupt flag
- #define **SDIO_CCCR_SUPPORT_8BIT_BUS** (1UL << 18U)

8 bit data bus flag
- #define **MMC_OCR_V170TO195_SHIFT** (7U)

The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define **MMC_OCR_V170TO195_MASK** (0x00000080U)

The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define **MMC_OCR_V200TO260_SHIFT** (8U)

The bit shift for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define **MMC_OCR_V200TO260_MASK** (0x00007F00U)

The bit mask for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define **MMC_OCR_V270TO360_SHIFT** (15U)

The bit shift for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define **MMC_OCR_V270TO360_MASK** (0x00FF8000U)

The bit mask for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define **MMC_OCR_ACCESS_MODE_SHIFT** (29U)

The bit shift for ACCESS MODE field in OCR.
- #define **MMC_OCR_ACCESS_MODE_MASK** (0x60000000U)

The bit mask for ACCESS MODE field in OCR.
- #define **MMC_OCR_BUSY_SHIFT** (31U)

The bit shift for BUSY field in OCR.
- #define **MMC_OCR_BUSY_MASK** (1U << MMC_OCR_BUSY_SHIFT)

The bit mask for BUSY field in OCR.
- #define **MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT** (0U)

The bit shift for FREQUENCY UNIT field in TRANSFER SPEED(TRAN-SPEED in Extended CSD)
- #define **MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK** (0x07U)

The bit mask for FREQUENCY UNIT in TRANSFER SPEED.
- #define **MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT** (3U)

The bit shift for MULTIPLIER field in TRANSFER SPEED.
- #define **MMC_TRANSFER_SPEED_MULTIPLIER_MASK** (0x78U)

The bit mask for MULTIPLIER field in TRANSFER SPEED.
- #define **READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(CSD)** (((CSD).transferSpeed) & **MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK**) >> **MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT**)

Read the value of FREQUENCY UNIT in TRANSFER SPEED.
- #define **READ_MMC_TRANSFER_SPEED_MULTIPLIER(CSD)** (((CSD).transferSpeed) & **MMC_TRANSFER_SPEED_MULTIPLIER_MASK**) >> **MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT**)

Read the value of MULTIPLIER filed in TRANSFER SPEED.
- #define **MMC_POWER_CLASS_4BIT_MASK** (0x0FU)

The power class value bit mask when bus in 4 bit mode.
- #define **MMC_POWER_CLASS_8BIT_MASK** (0xF0U)

The power class current value bit mask when bus in 8 bit mode.
- #define **MMC_CACHE_CONTROL_ENABLE** (1U)

- `mmc cache control enable`
- `#define MMC_CACHE_TRIGGER_FLUSH (1U)`
- mmc cache flush*
- `#define MMC_DATA_BUS_WIDTH_TYPE_NUMBER (3U)`
The number of data bus width type.
- `#define MMC_PARTITION_CONFIG_PARTITION_ACCESS_SHIFT (0U)`
The bit shift for PARTITION ACCESS filed in BOOT CONFIG (BOOT_CONFIG in Extend CSD).
- `#define MMC_PARTITION_CONFIG_PARTITION_ACCESS_MASK (0x00000007U)`
The bit mask for PARTITION ACCESS field in BOOT CONFIG.
- `#define MMC_PARTITION_CONFIG_PARTITION_ENABLE_SHIFT (3U)`
The bit shift for PARTITION ENABLE field in BOOT CONFIG.
- `#define MMC_PARTITION_CONFIG_PARTITION_ENABLE_MASK (0x00000038U)`
The bit mask for PARTITION ENABLE field in BOOT CONFIG.
- `#define MMC_PARTITION_CONFIG_BOOT_ACK_SHIFT (6U)`
The bit shift for ACK field in BOOT CONFIG.
- `#define MMC_PARTITION_CONFIG_BOOT_ACK_MASK (0x00000040U)`
The bit mask for ACK field in BOOT CONFIG.
- `#define MMC_BOOT_BUS_CONDITION_BUS_WIDTH_SHIFT (0U)`
The bit shift for BOOT BUS WIDTH field in BOOT CONFIG.
- `#define MMC_BOOT_BUS_CONDITION_BUS_WIDTH_MASK (3U)`
The bit mask for BOOT BUS WIDTH field in BOOT CONFIG.
- `#define MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_SHIFT (2U)`
The bit shift for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- `#define MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_MASK (4U)`
The bit mask for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- `#define MMC_BOOT_BUS_CONDITION_BOOT_MODE_SHIFT (3U)`
The bit shift for BOOT MODE field in BOOT CONFIG.
- `#define MMC_BOOT_BUS_CONDITION_BOOT_MODE_MASK (0x18U)`
The bit mask for BOOT MODE field in BOOT CONFIG.
- `#define MMC_EXTENDED_CSD_BYTES (512U)`
The length of Extended CSD register, unit as bytes.
- `#define MMC_DEFAULT_RELATIVE_ADDRESS (2UL)`
MMC card default relative address.
- `#define SD_PRODUCT_NAME_BYTES (5U)`
SD card product name length united as bytes.
- `#define SD_AU_START_VALUE (1U)`
SD AU start value.
- `#define SD_UHS_AU_START_VALUE (7U)`
SD UHS AU start value.
- `#define SD_TRANSFER_SPEED_RATE_UNIT_SHIFT (0U)`
The bit shift for RATE UNIT field in TRANSFER SPEED.
- `#define SD_TRANSFER_SPEED_RATE_UNIT_MASK (0x07U)`
The bit mask for RATE UNIT field in TRANSFER SPEED.
- `#define SD_TRANSFER_SPEED_TIME_VALUE_SHIFT (2U)`
The bit shift for TIME VALUE field in TRANSFER SPEED.
- `#define SD_TRANSFER_SPEED_TIME_VALUE_MASK (0x78U)`
The bit mask for TIME VALUE field in TRANSFER SPEED.
- `#define SD_RD_TRANSFER_SPEED_RATE_UNIT(x) (((x.transferSpeed) & SD_TRANSFER_SPEED_RATE_UNIT_MASK) >> SD_TRANSFER_SPEED_RATE_UNIT_SHIFT)`
Read the value of FREQUENCY UNIT in TRANSFER SPEED field.
- `#define SD_RD_TRANSFER_SPEED_TIME_VALUE(x) (((x.transferSpeed) & SD_TRANSFER-`

`_SPEED_TIME_VALUE_MASK) >> SD_TRANSFER_SPEED_TIME_VALUE_SHIFT)`

Read the value of TIME VALUE in TRANSFER SPEED field.

- `#define MMC_PRODUCT_NAME_BYTES (6U)`
MMC card product name length united as bytes.
- `#define MMC_SWITCH_COMMAND_SET_SHIFT (0U)`
The bit shift for COMMAND SET field in SWITCH command.
- `#define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)`
The bit mask for COMMAND set field in SWITCH command.
- `#define MMC_SWITCH_VALUE_SHIFT (8U)`
The bit shift for VALUE field in SWITCH command.
- `#define MMC_SWITCH_VALUE_MASK (0x0000FF00U)`
The bit mask for VALUE field in SWITCH command.
- `#define MMC_SWITCH_BYTE_INDEX_SHIFT (16U)`
The bit shift for BYTE INDEX field in SWITCH command.
- `#define MMC_SWITCH_BYTE_INDEX_MASK (0x00FF0000U)`
The bit mask for BYTE INDEX field in SWITCH command.
- `#define MMC_SWITCH_ACCESS_MODE_SHIFT (24U)`
The bit shift for ACCESS MODE field in SWITCH command.
- `#define MMC_SWITCH_ACCESS_MODE_MASK (0x03000000U)`
The bit mask for ACCESS MODE field in SWITCH command.

Typedefs

- `typedef void(* sd_cd_t)(bool isInserted, void *userData)`
card detect application callback definition
- `typedef bool(* sd_cd_status_t)(void)`
card detect status
- `typedef void(* sd_io_voltage_func_t)(sdmmc_operation_voltage_t voltage)`
card switch voltage function pointer
- `typedef void(* sd_pwr_t)(bool enable)`
card power control function pointer
- `typedef void(* sd_io_strength_t)(uint32_t busFreq)`
card io strength control
- `typedef void(* sdio_int_t)(void *userData)`
card interrupt function pointer

Enumerations

- enum {

kStatus_SDMMC_NotSupportYet = MAKE_STATUS(kStatusGroup_SDMMC, 0U),

kStatus_SDMMC_TransferFailed = MAKE_STATUS(kStatusGroup_SDMMC, 1U),

kStatus_SDMMC_SetCardBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDMMC, 2U),

kStatus_SDMMC_HostNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 3U),

kStatus_SDMMC_CardNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 4U),

kStatus_SDMMC_AllSendCidFailed = MAKE_STATUS(kStatusGroup_SDMMC, 5U),

kStatus_SDMMC_SendRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 6U),

kStatus_SDMMC_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 7U),

kStatus_SDMMC_SelectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 8U),

kStatus_SDMMC_SendScrFailed = MAKE_STATUS(kStatusGroup_SDMMC, 9U),

kStatus_SDMMC_SetDataBusWidthFailed = MAKE_STATUS(kStatusGroup_SDMMC, 10U),

kStatus_SDMMC_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 11U),

kStatus_SDMMC_HandShakeOperationConditionFailed,

kStatus_SDMMC_SendApplicationCommandFailed,

kStatus_SDMMC_SwitchFailed = MAKE_STATUS(kStatusGroup_SDMMC, 14U),

kStatus_SDMMC_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDMMC, 15U),

kStatus_SDMMC_WaitWriteCompleteFailed = MAKE_STATUS(kStatusGroup_SDMMC, 16U),

kStatus_SDMMC_SetBlockCountFailed = MAKE_STATUS(kStatusGroup_SDMMC, 17U),

kStatus_SDMMC_SetRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 18U),

kStatus_SDMMC_SwitchBusTimingFailed = MAKE_STATUS(kStatusGroup_SDMMC, 19U),

kStatus_SDMMC_SendExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 20U),

kStatus_SDMMC_ConfigureBootFailed = MAKE_STATUS(kStatusGroup_SDMMC, 21U),

kStatus_SDMMC_ConfigureExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 22-U),

kStatus_SDMMC_EnableHighCapacityEraseFailed,

kStatus_SDMMC_SendTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 24U),

kStatus_SDMMC_ReceiveTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 25U),

kStatus_SDMMC_SDIO_ResponseError = MAKE_STATUS(kStatusGroup_SDMMC, 26U),

kStatus_SDMMC_SDIO_InvalidArgument,

kStatus_SDMMC_SDIO_SendOperationConditionFail,

kStatus_SDMMC_InvalidVoltage = MAKE_STATUS(kStatusGroup_SDMMC, 29U),

kStatus_SDMMC_SDIO_SwitchHighSpeedFail = MAKE_STATUS(kStatusGroup_SDMMC, 30-U),

kStatus_SDMMC_SDIO_ReadCISFail = MAKE_STATUS(kStatusGroup_SDMMC, 31U),

kStatus_SDMMC_SDIO_InvalidCard = MAKE_STATUS(kStatusGroup_SDMMC, 32U),

kStatus_SDMMC_TuningFail = MAKE_STATUS(kStatusGroup_SDMMC, 33U),

kStatus_SDMMC_SwitchVoltageFail = MAKE_STATUS(kStatusGroup_SDMMC, 34U),

kStatus_SDMMC_SwitchVoltage18VFail33VSuccess = MAKE_STATUS(kStatusGroup_SDMM-

C, 35U),

```
kStatus_SDMMC_ReTuningRequest = MAKE_STATUS(kStatusGroup_SDMMC, 36U),
kStatus_SDMMC_SetDriverStrengthFail = MAKE_STATUS(kStatusGroup_SDMMC, 37U),
kStatus_SDMMC_SetPowerClassFail = MAKE_STATUS(kStatusGroup_SDMMC, 38U),
kStatus_SDMMC_HostNotReady = MAKE_STATUS(kStatusGroup_SDMMC, 39U),
kStatus_SDMMC_CardDetectFailed = MAKE_STATUS(kStatusGroup_SDMMC, 40U),
kStatus_SDMMC_AuSizeNotSetProperly = MAKE_STATUS(kStatusGroup_SDMMC, 41U),
kStatus_SDMMC_PollingCardIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 42U),
kStatus_SDMMC_DeselectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 43U),
kStatus_SDMMC_CardStatusIdle = MAKE_STATUS(kStatusGroup_SDMMC, 44U),
kStatus_SDMMC_CardStatusBusy = MAKE_STATUS(kStatusGroup_SDMMC, 45U),
kStatus_SDMMC_CardInitFailed = MAKE_STATUS(kStatusGroup_SDMMC, 46U) }
```

SD/MMC card API's running status.

- enum {

kSDMMC_SignalLineCmd = 1U,

kSDMMC_SignalLineData0 = 2U,

kSDMMC_SignalLineData1 = 4U,

kSDMMC_SignalLineData2 = 8U,

kSDMMC_SignalLineData3 = 16U,

kSDMMC_SignalLineData4 = 32U,

kSDMMC_SignalLineData5 = 64U,

kSDMMC_SignalLineData6 = 128U,

kSDMMC_SignalLineData7 = 256U }

sdmmc signal line

- enum **sdmmc_operation_voltage_t** {

kSDMMC_OperationVoltageNone = 0U,

kSDMMC_OperationVoltage330V = 1U,

kSDMMC_OperationVoltage300V = 2U,

kSDMMC_OperationVoltage180V = 3U }

card operation voltage

- enum {

kSDMMC_BusWidth1Bit = 0U,

kSDMMC_BusWidth4Bit = 1U,

kSDMMC_BusWidth8Bit = 2U }

card bus width

- enum { **kSDMMC_Support8BitWidth** = 1U }

sdmmc capability flag

- enum {

kSDMMC_DataPacketFormatLSBFirst,

kSDMMC_DataPacketFormatMSBFirst }

@ brief sdmmc data packet format

- enum **sd_detect_card_type_t** {

kSD_DetectCardByGpioCD,

kSD_DetectCardByHostCD,

kSD_DetectCardByHostDATA3 }

sd card detect type

- enum {

 kSD_Inserted = 1U,

 kSD_Removed = 0U }

 @ brief SD card detect status
- enum {

 kSD_DAT3PullDown = 0U,

 kSD_DAT3PullUp = 1U }

 @ brief SD card detect status
- enum **sd_io_voltage_ctrl_type_t** {

 kSD_IOVoltageCtrlNotSupport = 0U,

 kSD_IOVoltageCtrlByGpio = 2U }

 io voltage control type
- enum {

 kSDMMC_R1OutOfRangeFlag = 31,

 kSDMMC_R1AddressErrorFlag = 30,

 kSDMMC_R1BlockLengthErrorFlag = 29,

 kSDMMC_R1EraseSequenceErrorFlag = 28,

 kSDMMC_R1EraseParameterErrorFlag = 27,

 kSDMMC_R1WriteProtectViolationFlag = 26,

 kSDMMC_R1CardIsLockedFlag = 25,

 kSDMMC_R1LockUnlockFailedFlag = 24,

 kSDMMC_R1CommandCrcErrorFlag = 23,

 kSDMMC_R1IllegalCommandFlag = 22,

 kSDMMC_R1CardEccFailedFlag = 21,

 kSDMMC_R1CardControllerErrorFlag = 20,

 kSDMMC_R1ErrorFlag = 19,

 kSDMMC_R1CidCsdOverwriteFlag = 16,

 kSDMMC_R1WriteProtectEraseSkipFlag = 15,

 kSDMMC_R1CardEccDisabledFlag = 14,

 kSDMMC_R1EraseResetFlag = 13,

 kSDMMC_R1ReadyForDataFlag = 8,

 kSDMMC_R1SwitchErrorFlag = 7,

 kSDMMC_R1ApplicationCommandFlag = 5,

 kSDMMC_R1AuthenticationSequenceErrorFlag = 3 }

 Card status bit in R1.
- enum **sdmmc_r1_current_state_t** {

 kSDMMC_R1StateIdle = 0U,

 kSDMMC_R1StateReady = 1U,

 kSDMMC_R1StateIdentify = 2U,

 kSDMMC_R1StateStandby = 3U,

 kSDMMC_R1StateTransfer = 4U,

 kSDMMC_R1StateSendData = 5U,

 kSDMMC_R1StateReceiveData = 6U,

 kSDMMC_R1StateProgram = 7U,

 kSDMMC_R1StateDisconnect = 8U }

 CURRENT_STATE filed in R1.

- enum {

 kSDSPI_R1InIdleStateFlag = (1U << 0U),

 kSDSPI_R1EraseResetFlag = (1U << 1U),

 kSDSPI_R1IllegalCommandFlag = (1U << 2U),

 kSDSPI_R1CommandCrcErrorFlag = (1U << 3U),

 kSDSPI_R1EraseSequenceErrorFlag = (1U << 4U),

 kSDSPI_R1AddressErrorFlag = (1U << 5U),

 kSDSPI_R1ParameterErrorFlag = (1U << 6U) }

Error bit in SPI mode R1.

- enum {

 kSDSPI_R2CardLockedFlag = (1U << 0U),

 kSDSPI_R2WriteProtectEraseSkip = (1U << 1U),

 kSDSPI_R2LockUnlockFailed = (1U << 1U),

 kSDSPI_R2ErrorFlag = (1U << 2U),

 kSDSPI_R2CardControllerErrorFlag = (1U << 3U),

 kSDSPI_R2CardEccFailedFlag = (1U << 4U),

 kSDSPI_R2WriteProtectViolationFlag = (1U << 5U),

 kSDSPI_R2EraseParameterErrorFlag = (1U << 6U),

 kSDSPI_R2OutOfRangeFlag = (1U << 7U),

 kSDSPI_R2CsdOverwriteFlag = (1U << 7U) }

Error bit in SPI mode R2.

- enum {

 kSDSPI_DataErrorTokenError = (1U << 0U),

 kSDSPI_DataErrorTokenCardControllerError = (1U << 1U),

 kSDSPI_DataErrorTokenCardEccFailed = (1U << 2U),

 kSDSPI_DataErrorTokenOutOfRange = (1U << 3U) }

Data Error Token mask bit.

- enum **sdspi_data_token_t** {

 kSDSPI_DataTokenBlockRead = 0xFEU,

 kSDSPI_DataTokenSingleBlockWrite = 0xFEU,

 kSDSPI_DataTokenMultipleBlockWrite = 0xFCU,

 kSDSPI_DataTokenStopTransfer = 0xFDU }

Data Token.

- enum **sdspi_data_response_token_t** {

 kSDSPI_DataResponseTokenAccepted = 0x05U,

 kSDSPI_DataResponseTokenCrcError = 0x0BU,

 kSDSPI_DataResponseTokenWriteError = 0x0DU }

Data Response Token.

- enum **sd_command_t** {

 kSD_SendRelativeAddress = 3U,

 kSD_Switch = 6U,

 kSD_SendInterfaceCondition = 8U,

 kSD_VoltageSwitch = 11U,

 kSD_SpeedClassControl = 20U,

 kSD_EraseWriteBlockStart = 32U,

 kSD_EraseWriteBlockEnd = 33U,

- ```
kSD_SendTuningBlock = 19U }
 SD card individual commands.
```
- enum `sdspi_command_t` { `kSDSPI_CommandCrc` = 59U }

*SDSPI individual commands.*
- enum `sd_application_command_t` {
`kSD_ApplicationSetBusWdith` = 6U,
`kSD_ApplicationStatus` = 13U,
`kSD_ApplicationSendNumberWriteBlocks` = 22U,
`kSD_ApplicationSetWriteBlockEraseCount` = 23U,
`kSD_ApplicationSendOperationCondition` = 41U,
`kSD_ApplicationSetClearCardDetect` = 42U,
`kSD_ApplicationSendScr` = 51U }

*SD card individual application commands.*
- enum {
`kSDMMC_CommandClassBasic` = (1U << 0U),
`kSDMMC_CommandClassBlockRead` = (1U << 2U),
`kSDMMC_CommandClassBlockWrite` = (1U << 4U),
`kSDMMC_CommandClassErase` = (1U << 5U),
`kSDMMC_CommandClassWriteProtect` = (1U << 6U),
`kSDMMC_CommandClassLockCard` = (1U << 7U),
`kSDMMC_CommandClassApplicationSpecific` = (1U << 8U),
`kSDMMC_CommandClassInputOutputMode` = (1U << 9U),
`kSDMMC_CommandClassSwitch` = (1U << 10U) }

*SD card command class.*
- enum {
`kSD_OcrPowerUpBusyFlag` = 31,
`kSD_OcrHostCapacitySupportFlag` = 30,
`kSD_OcrCardCapacitySupportFlag` = `kSD_OcrHostCapacitySupportFlag`,
`kSD_OcrSwitch18RequestFlag` = 24,
`kSD_OcrSwitch18AcceptFlag` = `kSD_OcrSwitch18RequestFlag`,
`kSD_OcrVdd27_28Flag` = 15,
`kSD_OcrVdd28_29Flag` = 16,
`kSD_OcrVdd29_30Flag` = 17,
`kSD_OcrVdd30_31Flag` = 18,
`kSD_OcrVdd31_32Flag` = 19,
`kSD_OcrVdd32_33Flag` = 20,
`kSD_OcrVdd33_34Flag` = 21,
`kSD_OcrVdd34_35Flag` = 22,
`kSD_OcrVdd35_36Flag` = 23 } }

*OCR register in SD card.*
- enum {
`kSD_SpecificationVersion1_0` = (1U << 0U),
`kSD_SpecificationVersion1_1` = (1U << 1U),
`kSD_SpecificationVersion2_0` = (1U << 2U),
`kSD_SpecificationVersion3_0` = (1U << 3U) }

*SD card specification version number.*

- enum `sd_switch_mode_t` {
   
  `kSD_SwitchCheck` = 0U,
   
  `kSD_SwitchSet` = 1U }
   
    *SD card switch mode.*
- enum {
   
  `kSD_CsdReadBlockPartialFlag` = (1U << 0U),
   
  `kSD_CsdWriteBlockMisalignFlag` = (1U << 1U),
   
  `kSD_CsdReadBlockMisalignFlag` = (1U << 2U),
   
  `kSD_CsdDsrImplementedFlag` = (1U << 3U),
   
  `kSD_CsdEraseBlockEnabledFlag` = (1U << 4U),
   
  `kSD_CsdWriteProtectGroupEnabledFlag` = (1U << 5U),
   
  `kSD_CsdWriteBlockPartialFlag` = (1U << 6U),
   
  `kSD_CsdFileFormatGroupFlag` = (1U << 7U),
   
  `kSD_CsdCopyFlag` = (1U << 8U),
   
  `kSD_CsdPermanentWriteProtectFlag` = (1U << 9U),
   
  `kSD_CsdTemporaryWriteProtectFlag` = (1U << 10U) }
   
    *SD card CSD register flags.*
- enum {
   
  `kSD_ScrDataStatusAfterErase` = (1U << 0U),
   
  `kSD_ScrSdSpecification3` = (1U << 1U) }
   
    *SD card SCR register flags.*
- enum {
   
  `kSD_FunctionSDR12Deafult` = 0U,
   
  `kSD_FunctionSDR25HighSpeed` = 1U,
   
  `kSD_FunctionSDR50` = 2U,
   
  `kSD_FunctionSDR104` = 3U,
   
  `kSD_FunctionDDR50` = 4U }
   
    *SD timing function number.*
- enum {
   
  `kSD_GroupTimingMode` = 0U,
   
  `kSD_GroupCommandSystem` = 1U,
   
  `kSD_GroupDriverStrength` = 2U,
   
  `kSD_GroupCurrentLimit` = 3U }  
  
    *SD group number.*
- enum `sd_timing_mode_t` {
   
  `kSD_TimingSDR12DefaultMode` = 0U,
   
  `kSD_TimingSDR25HighSpeedMode` = 1U,
   
  `kSD_TimingSDR50Mode` = 2U,
   
  `kSD_TimingSDR104Mode` = 3U,
   
  `kSD_TimingDDR50Mode` = 4U }  
  
    *SD card timing mode flags.*
- enum `sd_driver_strength_t` {
   
  `kSD_DriverStrengthTypeB` = 0U,
   
  `kSD_DriverStrengthTypeA` = 1U,
   
  `kSD_DriverStrengthTypeC` = 2U,
   
  `kSD_DriverStrengthTypeD` = 3U }

- SD card driver strength.*
- enum `sd_max_current_t` {  
  `kSD_CurrentLimit200MA` = 0U,  
  `kSD_CurrentLimit400MA` = 1U,  
  `kSD_CurrentLimit600MA` = 2U,  
  `kSD_CurrentLimit800MA` = 3U }
- SD card current limit.*
- enum `sdmmc_command_t` {  
  `kSDMMC_GoIdleState` = 0U,  
  `kSDMMC_AllSendCid` = 2U,  
  `kSDMMC_SetDsr` = 4U,  
  `kSDMMC_SelectCard` = 7U,  
  `kSDMMC_SendCsd` = 9U,  
  `kSDMMC_SendCid` = 10U,  
  `kSDMMC_StopTransmission` = 12U,  
  `kSDMMC_SendStatus` = 13U,  
  `kSDMMC_GoInactiveState` = 15U,  
  `kSDMMC_SetBlockLength` = 16U,  
  `kSDMMC_ReadSingleBlock` = 17U,  
  `kSDMMC_ReadMultipleBlock` = 18U,  
  `kSDMMC_SetBlockCount` = 23U,  
  `kSDMMC_WriteSingleBlock` = 24U,  
  `kSDMMC_WriteMultipleBlock` = 25U,  
  `kSDMMC_ProgramCsd` = 27U,  
  `kSDMMC_SetWriteProtect` = 28U,  
  `kSDMMC_ClearWriteProtect` = 29U,  
  `kSDMMC_SendWriteProtect` = 30U,  
  `kSDMMC_Erase` = 38U,  
  `kSDMMC_LockUnlock` = 42U,  
  `kSDMMC_ApplicationCommand` = 55U,  
  `kSDMMC_GeneralCommand` = 56U,  
  `kSDMMC_ReadOcr` = 58U }
- SD/MMC card common commands.*
- enum {

```

kSDIO_RegCCCRSdioVer = 0x00U,
kSDIO_RegSDVersion = 0x01U,
kSDIO_RegIOEnable = 0x02U,
kSDIO_RegIOReady = 0x03U,
kSDIO_RegIOIntEnable = 0x04U,
kSDIO_RegIOIntPending = 0x05U,
kSDIO_RegIOAbort = 0x06U,
kSDIO_RegBusInterface = 0x07U,
kSDIO_RegCardCapability = 0x08U,
kSDIO_RegCommonCISPointer = 0x09U,
kSDIO_RegBusSuspend = 0x0C,
kSDIO_RegFunctionSelect = 0x0DU,
kSDIO_RegExecutionFlag = 0x0EU,
kSDIO_RegReadyFlag = 0x0FU,
kSDIO_RegFN0BlockSizeLow = 0x10U,
kSDIO_RegFN0BlockSizeHigh = 0x11U,
kSDIO_RegPowerControl = 0x12U,
kSDIO_RegBusSpeed = 0x13U,
kSDIO_RegUHSITimingSupport = 0x14U,
kSDIO_RegDriverStrength = 0x15U,
kSDIO_RegInterruptExtension = 0x16U }

 sdio card cccr register addr
• enum sdio_command_t {
 kSDIO_SendRelativeAddress = 3U,
 kSDIO_SendOperationCondition = 5U,
 kSDIO_SendInterfaceCondition = 8U,
 kSDIO_RWIODirect = 52U,
 kSDIO_RWIOExtended = 53U }

 sdio card individual commands
• enum sdio_func_num_t {
 kSDIO_FunctionNum0,
 kSDIO_FunctionNum1,
 kSDIO_FunctionNum2,
 kSDIO_FunctionNum3,
 kSDIO_FunctionNum4,
 kSDIO_FunctionNum5,
 kSDIO_FunctionNum6,
 kSDIO_FunctionNum7,
 kSDIO_FunctionMemory }

 sdio card individual commands
• enum {

```

```

kSDIO_StatusCmdCRCError = 0x8000U,
kSDIO_StatusIllegalCmd = 0x4000U,
kSDIO_StatusR6Error = 0x2000U,
kSDIO_StatusError = 0x0800U,
kSDIO_StatusFunctionNumError = 0x0200U,
kSDIO_StatusOutOfRange = 0x0100U }

 sdio command response flag
• enum {
 kSDIO_OcrPowerUpBusyFlag = 31,
 kSDIO_OcrIONumber = 28,
 kSDIO_OcrMemPresent = 27,
 kSDIO_OcrVdd20_21Flag = 8,
 kSDIO_OcrVdd21_22Flag = 9,
 kSDIO_OcrVdd22_23Flag = 10,
 kSDIO_OcrVdd23_24Flag = 11,
 kSDIO_OcrVdd24_25Flag = 12,
 kSDIO_OcrVdd25_26Flag = 13,
 kSDIO_OcrVdd26_27Flag = 14,
 kSDIO_OcrVdd27_28Flag = 15,
 kSDIO_OcrVdd28_29Flag = 16,
 kSDIO_OcrVdd29_30Flag = 17,
 kSDIO_OcrVdd30_31Flag = 18,
 kSDIO_OcrVdd31_32Flag = 19,
 kSDIO_OcrVdd32_33Flag = 20,
 kSDIO_OcrVdd33_34Flag = 21,
 kSDIO_OcrVdd34_35Flag = 22,
 kSDIO_OcrVdd35_36Flag = 23 }

 sdio operation condition flag
• enum {
 kSDIO_CCCRSupportDirectCmdDuringDataTrans = (1UL << 0U),
 kSDIO_CCCRSupportMultiBlock = (1UL << 1U),
 kSDIO_CCCRSupportReadWait = (1UL << 2U),
 kSDIO_CCCRSupportSuspendResume = (1UL << 3U),
 kSDIO_CCCRSupportIntDuring4BitDataTrans = (1UL << 4U),
 kSDIO_CCCRSupportLowSpeed1Bit = (1UL << 6U),
 kSDIO_CCCRSupportLowSpeed4Bit = (1UL << 7U),
 kSDIO_CCCRSupportMasterPowerControl = (1UL << 8U),
 kSDIO_CCCRSupportHighSpeed = (1UL << 9U),
 kSDIO_CCCRSupportContinuousSPIInt = (1UL << 10U) }

 sdio capability flag
• enum {
 kSDIO_FBRSupportCSA = (1U << 0U),
 kSDIO_FBRSupportPowerSelection = (1U << 1U) }

 sdio fbr flag
• enum sdio_bus_width_t {

```

- ```

kSDIO_DataBus1Bit = 0x00U,
kSDIO_DataBus4Bit = 0X02U,
kSDIO_DataBus8Bit = 0X03U }
    
```

sdio bus width
- enum `mmc_command_t` {


```

kMMC_SendOperationCondition = 1U,
kMMC_SetRelativeAddress = 3U,
kMMC_SleepAwake = 5U,
kMMC_Switch = 6U,
kMMC_SendExtendedCsd = 8U,
kMMC_ReadDataUntilStop = 11U,
kMMC_BusTestRead = 14U,
kMMC_SendingBusTest = 19U,
kMMC_WriteDataUntilStop = 20U,
kMMC_SendTuningBlock = 21U,
kMMC_ProgramCid = 26U,
kMMC_EraseGroupStart = 35U,
kMMC_EraseGroupEnd = 36U,
kMMC_FastInputOutput = 39U,
kMMC_GoInterruptState = 40U }
    
```

MMC card individual commands.
- enum `mmc_classified_voltage_t` {


```

kMMC_ClassifiedVoltageHigh = 0U,
kMMC_ClassifiedVoltageDual = 1U }
    
```

MMC card classified as voltage range.
- enum `mmc_classified_density_t` { `kMMC_ClassifiedDensityWithin2GB` = 0U }
- enum `mmc_access_mode_t` {


```

kMMC_AccessModeByte = 0U,
kMMC_AccessModeSector = 2U }
    
```

MMC card access mode(Access mode in OCR).
- enum `mmc_voltage_window_t` {


```

kMMC_VoltageWindowNone = 0U,
kMMC_VoltageWindow120 = 0x01U,
kMMC_VoltageWindow170to195 = 0x02U,
kMMC_VoltageWindows270to360 = 0x1FFU }
    
```

MMC card voltage window(VDD voltage window in OCR).
- enum `mmc_csd_structure_version_t` {


```

kMMC_CsdStrucureVersion10 = 0U,
kMMC_CsdStrucureVersion11 = 1U,
kMMC_CsdStrucureVersion12 = 2U,
kMMC_CsdStrucureVersionInExtcsd = 3U }
    
```

CSD structure version(CSD_STRUCTURE in CSD).
- enum `mmc_specification_version_t` {

```
kMMC_SpecificationVersion0 = 0U,
kMMC_SpecificationVersion1 = 1U,
kMMC_SpecificationVersion2 = 2U,
kMMC_SpecificationVersion3 = 3U,
kMMC_SpecificationVersion4 = 4U }
```

MMC card specification version(SPEC_VERS in CSD).

- enum {

kMMC_ExtendedCsdRevision10 = 0U,
 kMMC_ExtendedCsdRevision11 = 1U,
 kMMC_ExtendedCsdRevision12 = 2U,
 kMMC_ExtendedCsdRevision13 = 3U,
 kMMC_ExtendedCsdRevision14 = 4U,
 kMMC_ExtendedCsdRevision15 = 5U,
 kMMC_ExtendedCsdRevision16 = 6U,
 kMMC_ExtendedCsdRevision17 = 7U }

MMC card Extended CSD fix version(EXT_CSD_REV in Extended CSD)

- enum **mmc_command_set_t** {

kMMC_CommandSetStandard = 0U,
 kMMC_CommandSet1 = 1U,
 kMMC_CommandSet2 = 2U,
 kMMC_CommandSet3 = 3U,
 kMMC_CommandSet4 = 4U }

MMC card command set(COMMAND_SET in Extended CSD)

- enum {

kMMC_SupportAlternateBoot = 1U,
 kMMC_SupportDDRBoot = 2U,
 kMMC_SupportHighSpeedBoot = 4U }

boot support(BOOT_INFO in Extended CSD)

- enum **mmc_high_speed_timing_t** {

kMMC_HighSpeedTimingNone = 0U,
 kMMC_HighSpeedTiming = 1U,
 kMMC_HighSpeed200Timing = 2U,
 kMMC_HighSpeed400Timing = 3U,
 kMMC_EnhanceHighSpeed400Timing = 4U }

MMC card high-speed timing(HS_TIMING in Extended CSD)

- enum **mmc_data_bus_width_t** {

kMMC_DataBusWidth1bit = 0U,
 kMMC_DataBusWidth4bit = 1U,
 kMMC_DataBusWidth8bit = 2U,
 kMMC_DataBusWidth4bitDDR = 5U,
 kMMC_DataBusWidth8bitDDR = 6U,
 kMMC_DataBusWidth8bitDDRSTROBE = 0x86U }

MMC card data bus width(BUS_WIDTH in Extended CSD)

- enum **mmc_boot_partition_enable_t** {

```
kMMC_BootPartitionEnableNot = 0U,
kMMC_BootPartitionEnablePartition1 = 1U,
kMMC_BootPartitionEnablePartition2 = 2U,
kMMC_BootPartitionEnableUserAera = 7U }
```

MMC card boot partition enabled(BOOT_PARTITION_ENABLE in Extended CSD)

- enum `mmc_boot_timing_mode_t` {


```
kMMC_BootModeSDRWithDefaultTiming = 0U,
kMMC_BootModeSDRWithHighSpeedTiming = 1U,
kMMC_BootModeDDRTiming = 2U }
```

boot mode configuration Note: HS200 & HS400 is not support during BOOT operation.
- enum `mmc_boot_partition_wp_t` {


```
kMMC_BootPartitionWPDisable = 0x50U,
kMMC_BootPartitionPwrWPToBothPartition,
kMMC_BootPartitionPermWPToBothPartition = 0x04U,
kMMC_BootPartitionPwrWPToPartition1 = (1U << 7U) | 1U,
kMMC_BootPartitionPwrWPToPartition2 = (1U << 7U) | 3U,
kMMC_BootPartitionPermWPToPartition1,
kMMC_BootPartitionPermWPToPartition2,
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2,
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 }
```

MMC card boot partition write protect configurations All the bits in BOOT_WP register, except the two R/W bits B_PERM_WP_DIS and B_PERM_WP_EN, shall only be written once per power cycle. The protection mode intended for both boot areas will be set with a single write.

- enum {


```
kMMC_BootPartitionNotProtected = 0U,
kMMC_BootPartitionPwrProtected = 1U,
kMMC_BootPartitionPermProtected = 2U }
```

MMC card boot partition write protect status.

- enum `mmc_access_partition_t` {


```
kMMC_AccessPartitionUserAera = 0U,
kMMC_AccessPartitionBoot1 = 1U,
kMMC_AccessPartitionBoot2 = 2U,
kMMC_AccessRPMB = 3U,
kMMC_AccessGeneralPurposePartition1 = 4U,
kMMC_AccessGeneralPurposePartition2 = 5U,
kMMC_AccessGeneralPurposePartition3 = 6U,
kMMC_AccessGeneralPurposePartition4 = 7U }
```

MMC card partition to be accessed(BOOT_PARTITION_ACCESS in Extended CSD)

- enum {

```

kMMC_CsdReadBlockPartialFlag = (1U << 0U),
kMMC_CsdWriteBlockMisalignFlag = (1U << 1U),
kMMC_CsdReadBlockMisalignFlag = (1U << 2U),
kMMC_CsdDsrImplementedFlag = (1U << 3U),
kMMC_CsdWriteProtectGroupEnabledFlag = (1U << 4U),
kMMC_CsdWriteBlockPartialFlag = (1U << 5U),
kMMC_ContentProtectApplicationFlag = (1U << 6U),
kMMC_CsdFileFormatGroupFlag = (1U << 7U),
kMMC_CsdCopyFlag = (1U << 8U),
kMMC_CsdPermanentWriteProtectFlag = (1U << 9U),
kMMC_CsdTemporaryWriteProtectFlag = (1U << 10U) }

```

MMC card CSD register flags.

- enum `mmc_extended_csd_access_mode_t` {


```

kMMC_ExtendedCsdAccessModeCommandSet = 0U,
kMMC_ExtendedCsdAccessModeSetBits = 1U,
kMMC_ExtendedCsdAccessModeClearBits = 2U,
kMMC_ExtendedCsdAccessModeWriteBits = 3U 
```

Extended CSD register access mode(Access mode in CMD6).
- enum `mmc_extended_csd_index_t` {


```

kMMC_ExtendedCsdIndexFlushCache = 32U,
kMMC_ExtendedCsdIndexCacheControl = 33U,
kMMC_ExtendedCsdIndexBootPartitionWP = 173U,
kMMC_ExtendedCsdIndexEraseGroupDefinition = 175U,
kMMC_ExtendedCsdIndexBootBusConditions = 177U,
kMMC_ExtendedCsdIndexBootConfigWP = 178U,
kMMC_ExtendedCsdIndexPartitionConfig = 179U,
kMMC_ExtendedCsdIndexBusWidth = 183U,
kMMC_ExtendedCsdIndexHighSpeedTiming = 185U,
kMMC_ExtendedCsdIndexPowerClass = 187U,
kMMC_ExtendedCsdIndexCommandSet = 191U 
```

EXT CSD byte index.
- enum {


```

kMMC_DriverStrength0 = 0U,
kMMC_DriverStrength1 = 1U,
kMMC_DriverStrength2 = 2U,
kMMC_DriverStrength3 = 3U,
kMMC_DriverStrength4 = 4U 
```

mmc driver strength
- enum `mmc_extended_csd_flags_t` {


```

kMMC_ExtCsdExtPartitionSupport = (1 << 0U),
kMMC_ExtCsdEnhancePartitionSupport = (1 << 1U),
kMMC_ExtCsdPartitioningSupport = (1 << 2U),
kMMC_ExtCsdPrgCIDCSDInDDRModeSupport = (1 << 3U),
kMMC_ExtCsdBKOpsSupport = (1 << 4U),
kMMC_ExtCsdDataTagSupport = (1 << 5U),
kMMC_ExtCsdModeOperationCodeSupport = (1 << 6U) 
```

- enum `mmc_boot_mode_t` {

 `kMMC_BootModeNormal` = 0U,

 `kMMC_BootModeAlternative` = 1U }

 MMC card boot mode.

common function

tuning pattern

- `status_t SDMMC_SelectCard (sdmmchost_t *host, uint32_t relativeAddress, bool isSelected)`

 Selects the card to put it into transfer state.
- `status_t SDMMC_SendApplicationCommand (sdmmchost_t *host, uint32_t relativeAddress)`

 Sends an application command.
- `status_t SDMMC_SetBlockCount (sdmmchost_t *host, uint32_t blockCount)`

 Sets the block count.
- `status_t SDMMC_GoIdle (sdmmchost_t *host)`

 Sets the card to be idle state.
- `status_t SDMMC_SetBlockSize (sdmmchost_t *host, uint32_t blockSize)`

 Sets data block size.
- `status_t SDMMC_SetCardInactive (sdmmchost_t *host)`

 Sets card to inactive status.

46.7.2 Data Structure Documentation

46.7.2.1 struct sd_detect_card_t

Data Fields

- `sd_detect_card_type_t type`

 card detect type
- `uint32_t cdDebounce_ms`

 card detect debounce delay ms
- `sd_cd_t callback`

 card inserted callback which is meaningful for interrupt case
- `sd_cd_status_t cardDetected`

 used to check sd cd status when card detect through GPIO
- `sd_dat3_pull_t dat3PullFunc`

 function pointer of DATA3 pull up/down
- `void * userData`

 user data

46.7.2.2 struct sd_io_voltage_t

Data Fields

- `sd_io_voltage_ctrl_type_t type`

- **sd_io_voltage_func_t func**
io voltage switch function

46.7.2.3 struct sd_usr_param_t

Data Fields

- **sd_pwr_t pwr**
power control configuration pointer
- **uint32_t powerOnDelayMS**
power on delay time
- **uint32_t powerOffDelayMS**
power off delay time
- **sd_io_strength_t ioStrength**
switch sd io strength
- **sd_io_voltage_t * ioVoltage**
switch io voltage
- **sd_detect_card_t * cd**
card detect
- **uint32_t maxFreq**
board support maximum frequency
- **uint32_t capability**
board capability flag

46.7.2.4 struct sdio_card_int_t

Data Fields

- **void * userData**
user data
- **sdio_int_t cardInterrupt**
card int call back

46.7.2.5 struct sdio_usr_param_t

Data Fields

- **sd_pwr_t pwr**
power control configuration pointer
- **uint32_t powerOnDelayMS**
power on delay time
- **uint32_t powerOffDelayMS**
power off delay time
- **sd_io_strength_t ioStrength**
switch sd io strength
- **sd_io_voltage_t * ioVoltage**
switch io voltage

- `sd_detect_card_t * cd`
card detect
- `sdio_card_int_t * sdioInt`
card int
- `uint32_t maxFreq`
board support maximum frequency
- `uint32_t capability`
board capability flag

46.7.2.6 struct `sdio_fbr_t`

Data Fields

- `uint8_t flags`
current io flags
- `uint8_t ioStdFunctionCode`
current io standard function code
- `uint8_t ioExtFunctionCode`
current io extended function code
- `uint32_t ioPointerToCIS`
current io pointer to CIS
- `uint32_t ioPointerToCSA`
current io pointer to CSA
- `uint16_t ioBlockSize`
current io block size

46.7.2.7 struct `sdio_common_cis_t`

Data Fields

- `uint16_t mID`
manufacturer code
- `uint16_t mInfo`
manufacturer information
- `uint8_t funcID`
function ID
- `uint16_t fn0MaxBlkSize`
function 0 max block size
- `uint8_t maxTransSpeed`
max data transfer speed for all function

46.7.2.8 struct `sdio_func_cis_t`

Data Fields

- `uint8_t funcID`
function ID
- `uint8_t funcInfo`

- *function info*
- **uint8_t ioVersion**
level of application specification this io support
- **uint32_t cardPSN**
product serial number
- **uint32_t ioCSASize**
available CSA size for io
- **uint8_t ioCSAProperty**
CSA property.
- **uint16_t ioMaxBlockSize**
io max transfer data size
- **uint32_t ioOCR**
io ioeration condition
- **uint8_t ioOPMinPwr**
min current in operation mode
- **uint8_t ioOPAvgPwr**
average current in operation mode
- **uint8_t ioOPMaxPwr**
max current in operation mode
- **uint8_t ioSBMinPwr**
min current in standby mode
- **uint8_t ioSBAvgPwr**
average current in standby mode
- **uint8_t ioSBMaxPwr**
max current in standby mode
- **uint16_t ioMinBandWidth**
io min transfer bandwidth
- **uint16_t ioOptimumBandWidth**
io optimum transfer bandwidth
- **uint16_t ioReadyTimeout**
timeout value from enalbe to ready
- **uint16_t ioHighCurrentAvgCurrent**
*the average peak current (mA)
when IO operating in high current mode*
- **uint16_t ioHighCurrentMaxCurrent**
*the max peak current (mA)
when IO operating in high current mode*
- **uint16_t ioLowCurrentAvgCurrent**
*the average peak current (mA)
when IO operating in lower current mode*
- **uint16_t ioLowCurrentMaxCurrent**
*the max peak current (mA)
when IO operating in lower current mode*

46.7.2.9 struct sd_status_t

Data Fields

- **uint8_t busWidth**
current buswidth

- `uint8_t secureMode`
secured mode
- `uint16_t cardType`
sdcard type
- `uint32_t protectedSize`
size of protected area
- `uint8_t speedClass`
speed class of card
- `uint8_t performanceMove`
Performance of move indicated by 1[MB/S]step.
- `uint8_t auSize`
size of AU
- `uint16_t eraseSize`
number of AUs to be erased at a time
- `uint8_t eraseTimeout`
timeout value for erasing areas specified by UNIT OF ERASE AU
- `uint8_t eraseOffset`
fixed offset value added to erase time
- `uint8_t uhsSpeedGrade`
speed grade for UHS mode
- `uint8_t uhsAuSize`
size of AU for UHS mode

46.7.2.10 struct sd_cid_t

Data Fields

- `uint8_t manufacturerID`
Manufacturer ID [127:120].
- `uint16_t applicationID`
OEM/Application ID [119:104].
- `uint8_t productName [SD_PRODUCT_NAME_BYTES]`
Product name [103:64].
- `uint8_t productVersion`
Product revision [63:56].
- `uint32_t productSerialNumber`
Product serial number [55:24].
- `uint16_t manufacturerData`
Manufacturing date [19:8].

46.7.2.11 struct sd_csd_t

Data Fields

- `uint8_t csdStructure`
CSD structure [127:126].
- `uint8_t dataReadAccessTime1`
Data read access-time-1 [119:112].
- `uint8_t dataReadAccessTime2`

- **uint8_t transferSpeed**
Maximum data transfer rate [103:96].
- **uint16_t cardCommandClass**
Card command classes [95:84].
- **uint8_t readBlockLength**
Maximum read data block length [83:80].
- **uint16_t flags**
Flags in _sd_csd_flag.
- **uint32_t deviceSize**
Device size [73:62].
- **uint8_t readCurrentVddMin**
Maximum read current at VDD min [61:59].
- **uint8_t readCurrentVddMax**
Maximum read current at VDD max [58:56].
- **uint8_t writeCurrentVddMin**
Maximum write current at VDD min [55:53].
- **uint8_t writeCurrentVddMax**
Maximum write current at VDD max [52:50].
- **uint8_t deviceSizeMultiplier**
Device size multiplier [49:47].
- **uint8_t eraseSectorSize**
Erase sector size [45:39].
- **uint8_t writeProtectGroupSize**
Write protect group size [38:32].
- **uint8_t writeSpeedFactor**
Write speed factor [28:26].
- **uint8_t writeBlockLength**
Maximum write data block length [25:22].
- **uint8_t fileFormat**
File format [11:10].

46.7.2.12 struct sd_scr_t

Data Fields

- **uint8_t scrStructure**
SCR Structure [63:60].
- **uint8_t sdSpecification**
SD memory card specification version [59:56].
- **uint16_t flags**
SCR flags in _sd_scr_flag.
- **uint8_t sdSecurity**
Security specification supported [54:52].
- **uint8_t sdBusWidths**
Data bus widths supported [51:48].
- **uint8_t extendedSecurity**
Extended security support [46:43].
- **uint8_t commandSupport**
Command support bits [33:32] 33-support CMD23, 32-support cmd20.

- `uint32_t reservedForManufacturer`
reserved for manufacturer usage [31:0]

46.7.2.13 struct mmc_cid_t

Data Fields

- `uint8_t manufacturerID`
Manufacturer ID.
- `uint16_t applicationID`
OEM/Application ID.
- `uint8_t productName [MMC_PRODUCT_NAME_BYTES]`
Product name.
- `uint8_t productVersion`
Product revision.
- `uint32_t productSerialNumber`
Product serial number.
- `uint8_t manufacturerData`
Manufacturing date.

46.7.2.14 struct mmc_csd_t

Data Fields

- `uint8_t csdStructureVersion`
CSD structure [127:126].
- `uint8_t systemSpecificationVersion`
System specification version [125:122].
- `uint8_t dataReadAccessTime1`
Data read access-time 1 [119:112].
- `uint8_t dataReadAccessTime2`
*Data read access-time 2 in CLOCK cycles (NSAC*100) [111:104].*
- `uint8_t transferSpeed`
Max.
- `uint16_t cardCommandClass`
card command classes [95:84]
- `uint8_t readBlockLength`
Max.
- `uint16_t flags`
Contain flags in _mmc_csd_flag.
- `uint16_t deviceSize`
Device size [73:62].
- `uint8_t readCurrentVddMin`
Max.
- `uint8_t readCurrentVddMax`
Max.
- `uint8_t writeCurrentVddMin`
Max.
- `uint8_t writeCurrentVddMax`

- `uint8_t deviceSizeMultiplier`
Device size multiplier [49:47].
- `uint8_t eraseGroupSize`
Erase group size [46:42].
- `uint8_t eraseGroupSizeMultiplier`
Erase group size multiplier [41:37].
- `uint8_t writeProtectGroupSize`
Write protect group size [36:32].
- `uint8_t defaultEcc`
Manufacturer default ECC [30:29].
- `uint8_t writeSpeedFactor`
Write speed factor [28:26].
- `uint8_t maxWriteBlockLength`
Max.
- `uint8_t fileFormat`
File format [11:10].
- `uint8_t eccCode`
ECC code [9:8].

Field Documentation

(1) `uint8_t mmc_csd_t::transferSpeed`

bus clock frequency [103:96]

(2) `uint8_t mmc_csd_t::readBlockLength`

read data block length [83:80]

(3) `uint8_t mmc_csd_t::readCurrentVddMin`

read current @ VDD min [61:59]

(4) `uint8_t mmc_csd_t::readCurrentVddMax`

read current @ VDD max [58:56]

(5) `uint8_t mmc_csd_t::writeCurrentVddMin`

write current @ VDD min [55:53]

(6) `uint8_t mmc_csd_t::writeCurrentVddMax`

write current @ VDD max [52:50]

(7) `uint8_t mmc_csd_t::maxWriteBlockLength`

write data block length [25:22]

46.7.2.15 struct mmc_extended_csd_t

Data Fields

- `uint8_t cacheCtrl`
 $<$ secure removal type[16]
- `uint8_t partitionAttribute`
 $<$ power off notification[34]
- `uint8_t userWP`
 $<$ max enhance area size [159-157]
- `uint8_t bootPartitionWP`
boot write protect register[173]
- `uint8_t bootWPStatus`
boot write protect status register[174]
- `uint8_t highDensityEraseGroupDefinition`
High-density erase group definition [175].
- `uint8_t bootDataBusConditions`
Boot bus conditions [177].
- `uint8_t bootConfigProtect`
Boot config protection [178].
- `uint8_t partitionConfig`
Boot configuration [179].
- `uint8_t eraseMemoryContent`
Erased memory content [181].
- `uint8_t dataBusWidth`
Data bus width mode [183].
- `uint8_t highSpeedTiming`
High-speed interface timing [185].
- `uint8_t powerClass`
Power class [187].
- `uint8_t commandSetRevision`
Command set revision [189].
- `uint8_t commandSet`
Command set [191].
- `uint8_t extendecCsdVersion`
Extended CSD revision [192].
- `uint8_t csdStructureVersion`
CSD structure version [194].
- `uint8_t cardType`
Card Type [196].
- `uint8_t ioDriverStrength`
IO driver strength [197].
- `uint8_t partitionSwitchTimeout`
 $<$ out of interrupt busy timing [198]
- `uint8_t powerClass52MHz195V`
Power Class for 52MHz @ 1.95V [200].
- `uint8_t powerClass26MHz195V`
Power Class for 26MHz @ 1.95V [201].
- `uint8_t powerClass52MHz360V`
Power Class for 52MHz @ 3.6V [202].
- `uint8_t powerClass26MHz360V`

- **Power Class for 26MHz @ 3.6V [203].**
- **uint8_t minimumReadPerformance4Bit26MHz**
Minimum Read Performance for 4bit at 26MHz [205].
- **uint8_t minimumWritePerformance4Bit26MHz**
Minimum Write Performance for 4bit at 26MHz [206].
- **uint8_t minimumReadPerformance8Bit26MHz4Bit52MHz**
Minimum read Performance for 8bit at 26MHz/4bit @ 52MHz [207].
- **uint8_t minimumWritePerformance8Bit26MHz4Bit52MHz**
Minimum Write Performance for 8bit at 26MHz/4bit @ 52MHz [208].
- **uint8_t minimumReadPerformance8Bit52MHz**
Minimum Read Performance for 8bit at 52MHz [209].
- **uint8_t minimumWritePerformance8Bit52MHz**
Minimum Write Performance for 8bit at 52MHz [210].
- **uint32_t sectorCount**
Sector Count [215:212].
- **uint8_t sleepAwakeTimeout**
< sleep notification timeout [216]
- **uint8_t sleepCurrentVCCQ**
< Production state awareness timeout [218]
- **uint8_t sleepCurrentVCC**
Sleep current (VCC) [220].
- **uint8_t highCapacityWriteProtectGroupSize**
High-capacity write protect group size [221].
- **uint8_t reliableWriteSectorCount**
Reliable write sector count [222].
- **uint8_t highCapacityEraseTimeout**
High-capacity erase timeout [223].
- **uint8_t highCapacityEraseUnitSize**
High-capacity erase unit size [224].
- **uint8_t accessSize**
Access size [225].
- **uint8_t minReadPerformance8bitAt52MHZDDR**
< secure trim multiplier[229]
- **uint8_t minWritePerformance8bitAt52MHZDDR**
Minimum write performance for 8bit at DDR 52MHZ[235].
- **uint8_t powerClass200MHZVCCQ130VVCC360V**
power class for 200MHZ, at VCCQ= 1.3V,VCC=3.6V[236]
- **uint8_t powerClass200MHZVCCQ195VVCC360V**
power class for 200MHZ, at VCCQ= 1.95V,VCC=3.6V[237]
- **uint8_t powerClass52MHZDDR195V**
power class for 52MHZ,DDR at Vcc 1.95V[238]
- **uint8_t powerClass52MHZDDR360V**
power class for 52MHZ,DDR at Vcc 3.6V[239]
- **uint32_t genericCMD6Timeout**
< 1st initialization time after partitioning[241]
- **uint32_t cacheSize**
cache size[252-249]
- **uint8_t powerClass200MHZDDR360V**
power class for 200MHZ, DDR at VCC=2.6V[253]
- **uint8_t extPartitionSupport**
< fw VERSION [261-254]

- `uint8_t supportedCommandSet`
< large unit size[495]

Field Documentation

(1) `uint8_t mmc_extended_csd_t::cacheCtrl`

*< product state awareness enablement[17]
< max preload data size[21-18]
< pre-load data size[25-22]
< FFU status [26]
< mode operation code[29]
< mode config [30] control to turn on/off cache[33]*

(2) `uint8_t mmc_extended_csd_t::partitionAttribute`

*< packed cmd fail index [35]
< packed cmd status[36]
< context configuration[51-37]
< extended partitions attribut[53-52]
< exception events status[55-54]
< exception events control[57-56]
< number of group to be released[58]
< class 6 command control[59]
< 1st initialization after disabling sector size emu[60]
< sector size[61]
< sector size emulation[62]
< native sector size[63]
< period wakeup [131]
< package case temperature is controlled[132]
< production state awareness[133]
< enhanced user data start addr [139-136]
< enhanced user data area size[142-140]
< general purpose partition size[154-143] partition attribute [156]*

(3) `uint8_t mmc_extended_csd_t::userWP`

< HPI management [161]

< write reliability parameter register[166]
< write reliability setting register[167]
< RPMB size multi [168]
< FW configuration[169] user write protect register[171]

(4) uint8_t mmc_extended_csd_t::partitionSwitchTimeout

partition switch timing [199]

(5) uint8_t mmc_extended_csd_t::sleepAwakeTimeout

Sleep/awake timeout [217]

(6) uint8_t mmc_extended_csd_t::sleepCurrentVCCQ

Sleep current (VCCQ) [219]

(7) uint8_t mmc_extended_csd_t::minReadPerformance8bitAt52MHZDDR

< secure erase multiplier[230]
< secure feature support[231]
< trim multiplier[232] Minimum read performance for 8bit at DDR 52MHZ[234]

(8) uint32_t mmc_extended_csd_t::genericCMD6Timeout

< correct prg sectors number[245-242]
< background operations status[246]
< power off notification timeout[247] generic CMD6 timeout[248]

(9) uint8_t mmc_extended_csd_t::extPartitionSupport

< device version[263-262]
< optimal trim size[264]
< optimal write size[265]
< optimal read size[266]
< pre EOL information[267]
< device life time estimation typeA[268]
< device life time estimation typeB[269]
< number of FW sectors correctly programmed[305-302]
< FFU argument[490-487]
< operation code timeout[491]

< support mode [493] extended partition attribute support[494]

(10) uint8_t mmc_extended_csd_t::supportedCommandSet

< context management capability[496]

< tag resource size[497]

< tag unit size[498]

< max packed write cmd[500]

< max packed read cmd[501]

< HPI feature[503] Supported Command Sets [504]

46.7.2.16 struct mmc_extended_csd_config_t

Data Fields

- **mmc_command_set_t commandSet**
Command set.
- **uint8_t ByteValue**
The value to set.
- **uint8_t ByteIndex**
The byte index in Extended CSD(mmc_extended_csd_index_t)
- **mmc_extended_csd_access_mode_t accessMode**
Access mode.

46.7.2.17 struct mmc_boot_config_t

Data Fields

- **mmc_boot_mode_t bootMode**
mmc boot mode
- **bool enableBootAck**
Enable boot ACK.
- **mmc_boot_partition_enable_t bootPartition**
Boot partition.
- **mmc_boot_timing_mode_t bootTimingMode**
boot mode
- **mmc_data_bus_width_t bootDataBusWidth**
Boot data bus width.
- **bool retainBootbusCondition**
If retain boot bus width and boot mode conditions.
- **bool pwrBootConfigProtection**
Disable the change of boot configuration register bits from at this point until next power cycle or next H/W reset operation
- **bool premBootConfigProtection**
Disable the change of boot configuration register bits permanently.
- **mmc_boot_partition_wp_t bootPartitionWP**

boot partition write protect configurations

46.7.3 Macro Definition Documentation

46.7.3.1 `#define SDMMC_LOG(format, ...)`

46.7.3.2 `#define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)`

46.7.3.3 `#define READ_MMC_TRANSFER_SPEED_MULTIPLIER(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)`

46.7.3.4 `#define MMC_EXTENDED_CSD_BYTES (512U)`

46.7.3.5 `#define SD_PRODUCT_NAME_BYTES (5U)`

46.7.3.6 `#define MMC_PRODUCT_NAME_BYTES (6U)`

46.7.3.7 `#define MMC_SWITCH_COMMAND_SET_SHIFT (0U)`

46.7.3.8 `#define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)`

46.7.4 Enumeration Type Documentation

46.7.4.1 anonymous enum

Enumerator

`kStatus_SDMMC_NotSupportYet` Haven't supported.

`kStatus_SDMMC_TransferFailed` Send command failed.

`kStatus_SDMMC_SetCardBlockSizeFailed` Set block size failed.

`kStatus_SDMMC_HostNotSupport` Host doesn't support.

`kStatus_SDMMC_CardNotSupport` Card doesn't support.

`kStatus_SDMMC_AllSendCidFailed` Send CID failed.

`kStatus_SDMMC_SendRelativeAddressFailed` Send relative address failed.

`kStatus_SDMMC_SendCsdFailed` Send CSD failed.

`kStatus_SDMMC_SelectCardFailed` Select card failed.

`kStatus_SDMMC_SendScrFailed` Send SCR failed.

`kStatus_SDMMC_SetDataBusWidthFailed` Set bus width failed.

`kStatus_SDMMC_GoIdleFailed` Go idle failed.

`kStatus_SDMMC_HandShakeOperationConditionFailed` Send Operation Condition failed.

`kStatus_SDMMC_SendApplicationCommandFailed` Send application command failed.

kStatus_SDMMC_SwitchFailed Switch command failed.
kStatus_SDMMC_StopTransmissionFailed Stop transmission failed.
kStatus_SDMMC_WaitWriteCompleteFailed Wait write complete failed.
kStatus_SDMMC_SetBlockCountFailed Set block count failed.
kStatus_SDMMC_SetRelativeAddressFailed Set relative address failed.
kStatus_SDMMC_SwitchBusTimingFailed Switch high speed failed.
kStatus_SDMMC_SendExtendedCsdFailed Send EXT_CSD failed.
kStatus_SDMMC_ConfigureBootFailed Configure boot failed.
kStatus_SDMMC_ConfigureExtendedCsdFailed Configure EXT_CSD failed.
kStatus_SDMMC_EnableHighCapacityEraseFailed Enable high capacity erase failed.
kStatus_SDMMC_SendTestPatternFailed Send test pattern failed.
kStatus_SDMMC_ReceiveTestPatternFailed Receive test pattern failed.
kStatus_SDMMC_SDIO_ResponseError sdio response error
kStatus_SDMMC_SDIO_InvalidArgument sdio invalid argument response error
kStatus_SDMMC_SDIO_SendOperationConditionFail sdio send operation condition fail
kStatus_SDMMC_InvalidVoltage invalild voltage
kStatus_SDMMC_SDIO_SwitchHighSpeedFail switch to high speed fail
kStatus_SDMMC_SDIO_ReadCISFail read CIS fail
kStatus_SDMMC_SDIO_InvalidCard invalild SDIO card
kStatus_SDMMC_TuningFail tuning fail
kStatus_SDMMC_SwitchVoltageFail switch voltage fail
kStatus_SDMMC_SwitchVoltage18VFail33VSuccess switch voltage fail
kStatus_SDMMC_ReTuningRequest retuning request
kStatus_SDMMC_SetDriverStrengthFail set driver strength fail
kStatus_SDMMC_SetPowerClassFail set power class fail
kStatus_SDMMC_HostNotReady host controller not ready
kStatus_SDMMC_CardDetectFailed card detect failed
kStatus_SDMMC_AuSizeNotSetProperly AU size not set properly.
kStatus_SDMMC_PollingCardIdleFailed polling card idle status failed
kStatus_SDMMC_DeselectCardFailed deselect card failed
kStatus_SDMMC_CardStatusIdle card idle
kStatus_SDMMC_CardStatusBusy card busy
kStatus_SDMMC_CardInitFailed card init failed

46.7.4.2 anonymous enum

Enumerator

<i>kSDMMC_SignalLineCmd</i>	cmd line
<i>kSDMMC_SignalLineData0</i>	data line
<i>kSDMMC_SignalLineData1</i>	data line
<i>kSDMMC_SignalLineData2</i>	data line
<i>kSDMMC_SignalLineData3</i>	data line
<i>kSDMMC_SignalLineData4</i>	data line
<i>kSDMMC_SignalLineData5</i>	data line

kSDMMC_SignalLineData6 data line
kSDMMC_SignalLineData7 data line

46.7.4.3 enum sdmmc_operation_voltage_t

Enumerator

kSDMMC_OperationVoltageNone indicate current voltage setting is not setting by suser
kSDMMC_OperationVoltage330V card operation voltage around 3.3v
kSDMMC_OperationVoltage300V card operation voltage around 3.0v
kSDMMC_OperationVoltage180V card operation voltage around 1.8v

46.7.4.4 anonymous enum

Enumerator

kSDMMC_BusWidth1Bit card bus 1 width
kSDMMC_BusWidth4Bit card bus 4 width
kSDMMC_BusWidth8Bit card bus 8 width

46.7.4.5 anonymous enum

Enumerator

kSDMMC_Support8BitWidth 8 bit data width capability

46.7.4.6 anonymous enum

Enumerator

kSDMMC_DataPacketFormatLSBFirst usual data packet format LSB first, MSB last
kSDMMC_DataPacketFormatMSBFirst Wide width data packet format MSB first, LSB last.

46.7.4.7 enum sd_detect_card_type_t

Enumerator

kSD_DetectCardByGpioCD sd card detect by CD pin through GPIO
kSD_DetectCardByHostCD sd card detect by CD pin through host
kSD_DetectCardByHostDATA3 sd card detect by DAT3 pin through host

46.7.4.8 anonymous enum

Enumerator

kSD_Inserted card is inserted

kSD_Removed card is removed

46.7.4.9 anonymous enum

Enumerator

kSD_DAT3PullDown data3 pull down

kSD_DAT3PullUp data3 pull up

46.7.4.10 enum sd_io_voltage_ctrl_type_t

Enumerator

kSD_IOVoltageCtrlNotSupport io voltage control not support

kSD_IOVoltageCtrlByGpio io voltage control by gpio

46.7.4.11 anonymous enum

Enumerator

kSDMMC_R1OutOfRangeFlag Out of range status bit.

kSDMMC_R1AddressErrorFlag Address error status bit.

kSDMMC_R1BlockLengthErrorFlag Block length error status bit.

kSDMMC_R1EraseSequenceErrorFlag Erase sequence error status bit.

kSDMMC_R1EraseParameterErrorFlag Erase parameter error status bit.

kSDMMC_R1WriteProtectViolationFlag Write protection violation status bit.

kSDMMC_R1CardIsLockedFlag Card locked status bit.

kSDMMC_R1LockUnlockFailedFlag lock/unlock error status bit

kSDMMC_R1CommandCrcErrorFlag CRC error status bit.

kSDMMC_R1IllegalCommandFlag Illegal command status bit.

kSDMMC_R1CardEccFailedFlag Card ecc error status bit.

kSDMMC_R1CardControllerErrorFlag Internal card controller error status bit.

kSDMMC_R1ErrorFlag A general or an unknown error status bit.

kSDMMC_R1CidCsdOverwriteFlag Cid/csd overwrite status bit.

kSDMMC_R1WriteProtectEraseSkipFlag Write protection erase skip status bit.

kSDMMC_R1CardEccDisabledFlag Card ecc disabled status bit.

kSDMMC_R1EraseResetFlag Erase reset status bit.

kSDMMC_R1ReadyForDataFlag Ready for data status bit.

kSDMMC_R1SwitchErrorFlag Switch error status bit.

kSDMMC_R1ApplicationCommandFlag Application command enabled status bit.

kSDMMC_R1AuthenticationSequenceErrorFlag error in the sequence of authentication process

46.7.4.12 enum sdmmc_r1_current_state_t

Enumerator

kSDMMC_R1StateIdle R1: current state: idle.
kSDMMC_R1StateReady R1: current state: ready.
kSDMMC_R1StateIdentify R1: current state: identification.
kSDMMC_R1StateStandby R1: current state: standby.
kSDMMC_R1StateTransfer R1: current state: transfer.
kSDMMC_R1StateSendData R1: current state: sending data.
kSDMMC_R1StateReceiveData R1: current state: receiving data.
kSDMMC_R1StateProgram R1: current state: programming.
kSDMMC_R1StateDisconnect R1: current state: disconnect.

46.7.4.13 anonymous enum

Enumerator

kSDSPI_R1InIdleStateFlag In idle state.
kSDSPI_R1EraseResetFlag Erase reset.
kSDSPI_R1IllegalCommandFlag Illegal command.
kSDSPI_R1CommandCrcErrorFlag Com crc error.
kSDSPI_R1EraseSequenceErrorFlag Erase sequence error.
kSDSPI_R1AddressErrorFlag Address error.
kSDSPI_R1ParameterErrorFlag Parameter error.

46.7.4.14 anonymous enum

Enumerator

kSDSPI_R2CardLockedFlag Card is locked.
kSDSPI_R2WriteProtectEraseSkip Write protect erase skip.
kSDSPI_R2LockUnlockFailed Lock/unlock command failed.
kSDSPI_R2ErrorFlag Unknown error.
kSDSPI_R2CardControllerErrorFlag Card controller error.
kSDSPI_R2CardEccFailedFlag Card ecc failed.
kSDSPI_R2WriteProtectViolationFlag Write protect violation.
kSDSPI_R2EraseParameterErrorFlag Erase parameter error.
kSDSPI_R2OutOfRangeFlag Out of range.
kSDSPI_R2CsdOverwriteFlag CSD overwrite.

46.7.4.15 anonymous enum

Enumerator

- kSDSPI_DataErrorTokenError* Data error.
- kSDSPI_DataErrorTokenCardControllerError* Card controller error.
- kSDSPI_DataErrorTokenCardEccFailed* Card ecc error.
- kSDSPI_DataErrorTokenOutOfRange* Out of range.

46.7.4.16 enum sdspi_data_token_t

Enumerator

- kSDSPI_DataTokenBlockRead* Single block read, multiple block read.
- kSDSPI_DataTokenSingleBlockWrite* Single block write.
- kSDSPI_DataTokenMultipleBlockWrite* Multiple block write.
- kSDSPI_DataTokenStopTransfer* Stop transmission.

46.7.4.17 enum sdspi_data_response_token_t

Enumerator

- kSDSPI_DataResponseTokenAccepted* Data accepted.
- kSDSPI_DataResponseTokenCrcError* Data rejected due to CRC error.
- kSDSPI_DataResponseTokenWriteError* Data rejected due to write error.

46.7.4.18 enum sd_command_t

Enumerator

- kSD_SendRelativeAddress* Send Relative Address.
- kSD_Switch* Switch Function.
- kSD_SendInterfaceCondition* Send Interface Condition.
- kSD_VoltageSwitch* Voltage Switch.
- kSD_SpeedClassControl* Speed Class control.
- kSD_EraseWriteBlockStart* Write Block Start.
- kSD_EraseWriteBlockEnd* Write Block End.
- kSD_SendTuningBlock* Send Tuning Block.

46.7.4.19 enum sdspi_command_t

Enumerator

- kSDSPI_CommandCrc* Command crc protection on/off.

46.7.4.20 enum sd_application_command_t

Enumerator

kSD_ApplicationSetBusWidth Set Bus Width.
kSD_ApplicationStatus Send SD status.
kSD_ApplicationSendNumberWriteBlocks Send Number Of Written Blocks.
kSD_ApplicationSetWriteBlockEraseCount Set Write Block Erase Count.
kSD_ApplicationSendOperationCondition Send Operation Condition.
kSD_ApplicationSetClearCardDetect Set Connnect/Disconnect pull up on detect pin.
kSD_ApplicationSendScr Send Scr.

46.7.4.21 anonymous enum

Enumerator

kSDMMC_CommandClassBasic Card command class 0.
kSDMMC_CommandClassBlockRead Card command class 2.
kSDMMC_CommandClassBlockWrite Card command class 4.
kSDMMC_CommandClassErase Card command class 5.
kSDMMC_CommandClassWriteProtect Card command class 6.
kSDMMC_CommandClassLockCard Card command class 7.
kSDMMC_CommandClassApplicationSpecific Card command class 8.
kSDMMC_CommandClassInputOutputMode Card command class 9.
kSDMMC_CommandClassSwitch Card command class 10.

46.7.4.22 anonymous enum

Enumerator

kSD_OcrPowerUpBusyFlag Power up busy status.
kSD_OcrHostCapacitySupportFlag Card capacity status.
kSD_OcrCardCapacitySupportFlag Card capacity status.
kSD_OcrSwitch18RequestFlag Switch to 1.8V request.
kSD_OcrSwitch18AcceptFlag Switch to 1.8V accepted.
kSD_OcrVdd27_28Flag VDD 2.7-2.8.
kSD_OcrVdd28_29Flag VDD 2.8-2.9.
kSD_OcrVdd29_30Flag VDD 2.9-3.0.
kSD_OcrVdd30_31Flag VDD 2.9-3.0.
kSD_OcrVdd31_32Flag VDD 3.0-3.1.
kSD_OcrVdd32_33Flag VDD 3.1-3.2.
kSD_OcrVdd33_34Flag VDD 3.2-3.3.
kSD_OcrVdd34_35Flag VDD 3.3-3.4.
kSD_OcrVdd35_36Flag VDD 3.4-3.5.

46.7.4.23 anonymous enum

Enumerator

kSD_SpecificationVersion1_0 SD card version 1.0-1.01.*kSD_SpecificationVersion1_1* SD card version 1.10.*kSD_SpecificationVersion2_0* SD card version 2.00.*kSD_SpecificationVersion3_0* SD card version 3.0.**46.7.4.24 enum sd_switch_mode_t**

Enumerator

kSD_SwitchCheck SD switch mode 0: check function.*kSD_SwitchSet* SD switch mode 1: set function.**46.7.4.25 anonymous enum**

Enumerator

kSD_CsdReadBlockPartialFlag Partial blocks for read allowed [79:79].*kSD_CsdWriteBlockMisalignFlag* Write block misalignment [78:78].*kSD_CsdReadBlockMisalignFlag* Read block misalignment [77:77].*kSD_CsdDsrImplementedFlag* DSR implemented [76:76].*kSD_CsdEraseBlockEnabledFlag* Erase single block enabled [46:46].*kSD_CsdWriteProtectGroupEnabledFlag* Write protect group enabled [31:31].*kSD_CsdWriteBlockPartialFlag* Partial blocks for write allowed [21:21].*kSD_CsdFileFormatGroupFlag* File format group [15:15].*kSD_CsdCopyFlag* Copy flag [14:14].*kSD_CsdPermanentWriteProtectFlag* Permanent write protection [13:13].*kSD_CsdTemporaryWriteProtectFlag* Temporary write protection [12:12].**46.7.4.26 anonymous enum**

Enumerator

kSD_ScrDataStatusAfterErase Data status after erases [55:55].*kSD_ScrSdSpecification3* Specification version 3.00 or higher [47:47].**46.7.4.27 anonymous enum**

Enumerator

kSD_FunctionSDR12Default SDR12 mode & default.*kSD_FunctionSDR25HighSpeed* SDR25 & high speed.

kSD_FunctionSDR50 SDR50 mode.
kSD_FunctionSDR104 SDR104 mode.
kSD_FunctionDDR50 DDR50 mode.

46.7.4.28 anonymous enum

Enumerator

kSD_GroupTimingMode access mode group
kSD_GroupCommandSystem command system group
kSD_GroupDriverStrength driver strength group
kSD_GroupCurrentLimit current limit group

46.7.4.29 enum sd_timing_mode_t

Enumerator

kSD_TimingSDR12DefaultMode Identification mode & SDR12.
kSD_TimingSDR25HighSpeedMode High speed mode & SDR25.
kSD_TimingSDR50Mode SDR50 mode.
kSD_TimingSDR104Mode SDR104 mode.
kSD_TimingDDR50Mode DDR50 mode.

46.7.4.30 enum sd_driver_strength_t

Enumerator

kSD_DriverStrengthTypeB default driver strength
kSD_DriverStrengthTypeA driver strength TYPE A
kSD_DriverStrengthTypeC driver strength TYPE C
kSD_DriverStrengthTypeD driver strength TYPE D

46.7.4.31 enum sd_max_current_t

Enumerator

kSD_CurrentLimit200MA default current limit
kSD_CurrentLimit400MA current limit to 400MA
kSD_CurrentLimit600MA current limit to 600MA
kSD_CurrentLimit800MA current limit to 800MA

46.7.4.32 enum sdmmc_command_t

Enumerator

kSDMMC_GoIdleState Go Idle State.
kSDMMC_AllSendCid All Send CID.
kSDMMC_SetDsr Set DSR.
kSDMMC_SelectCard Select Card.
kSDMMC_SendCsd Send CSD.
kSDMMC_SendCid Send CID.
kSDMMC_StopTransmission Stop Transmission.
kSDMMC_SendStatus Send Status.
kSDMMC_GoInactiveState Go Inactive State.
kSDMMC_SetBlockLength Set Block Length.
kSDMMC_ReadSingleBlock Read Single Block.
kSDMMC_ReadMultipleBlock Read Multiple Block.
kSDMMC_SetBlockCount Set Block Count.
kSDMMC_WriteSingleBlock Write Single Block.
kSDMMC_WriteMultipleBlock Write Multiple Block.
kSDMMC_ProgramCsd Program CSD.
kSDMMC_SetWriteProtect Set Write Protect.
kSDMMC_ClearWriteProtect Clear Write Protect.
kSDMMC_SendWriteProtect Send Write Protect.
kSDMMC_Erase Erase.
kSDMMC_LockUnlock Lock Unlock.
kSDMMC_ApplicationCommand Send Application Command.
kSDMMC_GeneralCommand General Purpose Command.
kSDMMC_ReadOcr Read OCR.

46.7.4.33 anonymous enum

Enumerator

kSDIO_RegCCCRSdioVer CCCR & SDIO version.
kSDIO_RegSDVersion SD version.
kSDIO_RegIOEnable io enable register
kSDIO_RegIOReady io ready register
kSDIO_RegIOIntEnable io interrupt enable register
kSDIO_RegIOIntPending io interrupt pending register
kSDIO_RegIOAbort io abort register
kSDIO_RegBusInterface bus interface register
kSDIO_RegCardCapability card capability register
kSDIO_RegCommonCISPointer common CIS pointer register
kSDIO_RegBusSuspend bus suspend register
kSDIO_RegFunctionSelect function select register
kSDIO_RegExecutionFlag execution flag register

kSDIO_RegReadyFlag ready flag register
kSDIO_RegFN0BlockSizeLow FN0 block size register.
kSDIO_RegFN0BlockSizeHigh FN0 block size register.
kSDIO_RegPowerControl power control register
kSDIO_RegBusSpeed bus speed register
kSDIO_RegUHSITimingSupport UHS-I timing support register.
kSDIO_RegDriverStrength Driver strength register.
kSDIO_RegInterruptExtension Interrupt extension register.

46.7.4.34 enum sdio_command_t

Enumerator

kSDIO_SendRelativeAddress send relative address
kSDIO_SendOperationCondition send operation condition
kSDIO_SendInterfaceCondition send interface condition
kSDIO_RWIODirect read/write IO direct command
kSDIO_RWIOExtended read/write IO extended command

46.7.4.35 enum sdio_func_num_t

Enumerator

kSDIO_FunctionNum0 sdio function0
kSDIO_FunctionNum1 sdio function1
kSDIO_FunctionNum2 sdio function2
kSDIO_FunctionNum3 sdio function3
kSDIO_FunctionNum4 sdio function4
kSDIO_FunctionNum5 sdio function5
kSDIO_FunctionNum6 sdio function6
kSDIO_FunctionNum7 sdio function7
kSDIO_FunctionMemory for combo card

46.7.4.36 anonymous enum

Enumerator

kSDIO_StatusCmdCRCError the CRC check of the previous cmd fail
kSDIO_StatusIllegalCmd cmd illegal for the card state
kSDIO_StatusR6Error special for R6 error status
kSDIO_StatusError A general or an unknown error occurred.
kSDIO_StatusFunctionNumError invalid function error
kSDIO_StatusOutOfRange cmd argument was out of the allowed range

46.7.4.37 anonymous enum

Enumerator

<i>kSDIO_OcrPowerUpBusyFlag</i>	Power up busy status.
<i>kSDIO_OcrIONumber</i>	number of IO function
<i>kSDIO_OcrMemPresent</i>	memory present flag
<i>kSDIO_OcrVdd20_21Flag</i>	VDD 2.0-2.1.
<i>kSDIO_OcrVdd21_22Flag</i>	VDD 2.1-2.2.
<i>kSDIO_OcrVdd22_23Flag</i>	VDD 2.2-2.3.
<i>kSDIO_OcrVdd23_24Flag</i>	VDD 2.3-2.4.
<i>kSDIO_OcrVdd24_25Flag</i>	VDD 2.4-2.5.
<i>kSDIO_OcrVdd25_26Flag</i>	VDD 2.5-2.6.
<i>kSDIO_OcrVdd26_27Flag</i>	VDD 2.6-2.7.
<i>kSDIO_OcrVdd27_28Flag</i>	VDD 2.7-2.8.
<i>kSDIO_OcrVdd28_29Flag</i>	VDD 2.8-2.9.
<i>kSDIO_OcrVdd29_30Flag</i>	VDD 2.9-3.0.
<i>kSDIO_OcrVdd30_31Flag</i>	VDD 2.9-3.0.
<i>kSDIO_OcrVdd31_32Flag</i>	VDD 3.0-3.1.
<i>kSDIO_OcrVdd32_33Flag</i>	VDD 3.1-3.2.
<i>kSDIO_OcrVdd33_34Flag</i>	VDD 3.2-3.3.
<i>kSDIO_OcrVdd34_35Flag</i>	VDD 3.3-3.4.
<i>kSDIO_OcrVdd35_36Flag</i>	VDD 3.4-3.5.

46.7.4.38 anonymous enum

Enumerator

<i>kSDIO_CCCRSupportDirectCmdDuringDataTrans</i>	support direct cmd during data transfer
<i>kSDIO_CCCRSupportMultiBlock</i>	support multi block mode
<i>kSDIO_CCCRSupportReadWait</i>	support read wait
<i>kSDIO_CCCRSupportSuspendResume</i>	support suspend resume
<i>kSDIO_CCCRSupportIntDuring4BitDataTrans</i>	support interrupt during 4-bit data transfer
<i>kSDIO_CCCRSupportLowSpeed1Bit</i>	support low speed 1bit mode
<i>kSDIO_CCCRSupportLowSpeed4Bit</i>	support low speed 4bit mode
<i>kSDIO_CCCRSupportMasterPowerControl</i>	support master power control
<i>kSDIO_CCCRSupportHighSpeed</i>	support high speed
<i>kSDIO_CCCRSupportContinuousSPIInt</i>	support continuous SPI interrupt

46.7.4.39 anonymous enum

Enumerator

<i>kSDIO_FBRSupportCSA</i>	function support CSA
<i>kSDIO_FBRSupportPowerSelection</i>	function support power selection

46.7.4.40 enum sdio_bus_width_t

Enumerator

kSDIO_DataBus1Bit 1 bit bus mode
kSDIO_DataBus4Bit 4 bit bus mode
kSDIO_DataBus8Bit 8 bit bus mode

46.7.4.41 enum mmc_command_t

Enumerator

kMMC_SendOperationCondition Send Operation Condition.
kMMC_SetRelativeAddress Set Relative Address.
kMMC_SleepAwake Sleep Awake.
kMMC_Switch Switch.
kMMC_SendExtendedCsd Send EXT_CSD.
kMMC_ReadDataUntilStop Read Data Until Stop.
kMMC_BusTestRead Test Read.
kMMC_SendingBusTest test bus width cmd
kMMC_WriteDataUntilStop Write Data Until Stop.
kMMC_SendTuningBlock MMC sending tuning block.
kMMC_ProgramCid Program CID.
kMMC_EraseGroupStart Erase Group Start.
kMMC_EraseGroupEnd Erase Group End.
kMMC_FastInputOutput Fast IO.
kMMC_GoInterruptState Go interrupt State.

46.7.4.42 enum mmc_classified_voltage_t

Enumerator

kMMC_ClassifiedVoltageHigh High-voltage MMC card.
kMMC_ClassifiedVoltageDual Dual-voltage MMC card.

46.7.4.43 enum mmc_classified_density_t

Enumerator

kMMC_ClassifiedDensityWithin2GB Density byte is less than or equal 2GB.

46.7.4.44 enum mmc_access_mode_t

Enumerator

kMMC_AccessModeByte The card should be accessed as byte.

kMMC_AccessModeSector The card should be accessed as sector.

46.7.4.45 enum mmc_voltage_window_t

Enumerator

kMMC_VoltageWindowNone voltage window is not define by user

kMMC_VoltageWindow120 Voltage window is 1.20V.

kMMC_VoltageWindow170to195 Voltage window is 1.70V to 1.95V.

kMMC_VoltageWindows270to360 Voltage window is 2.70V to 3.60V.

46.7.4.46 enum mmc_csd_structure_version_t

Enumerator

kMMC_CsdStrucureVersion10 CSD version No. 1.0

kMMC_CsdStrucureVersion11 CSD version No. 1.1

kMMC_CsdStrucureVersion12 CSD version No. 1.2

kMMC_CsdStrucureVersionInExtcsd Version coded in Extended CSD.

46.7.4.47 enum mmc_specification_version_t

Enumerator

kMMC_SpecificationVersion0 Allocated by MMCA.

kMMC_SpecificationVersion1 Allocated by MMCA.

kMMC_SpecificationVersion2 Allocated by MMCA.

kMMC_SpecificationVersion3 Allocated by MMCA.

kMMC_SpecificationVersion4 Version 4.1/4.2/4.3/4.41-4.5-4.51-5.0.

46.7.4.48 anonymous enum

Enumerator

kMMC_ExtendedCsdRevision10 Revision 1.0.

kMMC_ExtendedCsdRevision11 Revision 1.1.

kMMC_ExtendedCsdRevision12 Revision 1.2.

kMMC_ExtendedCsdRevision13 Revision 1.3 MMC4.3.

kMMC_ExtendedCsdRevision14 Revision 1.4 obsolete.

- kMMC_ExtendedCsdRevision15*** Revision 1.5 MMC4.41.
- kMMC_ExtendedCsdRevision16*** Revision 1.6 MMC4.5.
- kMMC_ExtendedCsdRevision17*** Revision 1.7 MMC5.0.

46.7.4.49 enum mmc_command_set_t

Enumerator

- kMMC_CommandSetStandard*** Standard MMC.
- kMMC_CommandSet1*** Command set 1.
- kMMC_CommandSet2*** Command set 2.
- kMMC_CommandSet3*** Command set 3.
- kMMC_CommandSet4*** Command set 4.

46.7.4.50 anonymous enum

Enumerator

- kMMC_SupportAlternateBoot*** support alternative boot mode
- kMMC_SupportDDRBoot*** support DDR boot mode
- kMMC_SupportHighSpeedBoot*** support high speed boot mode

46.7.4.51 enum mmc_high_speed_timing_t

Enumerator

- kMMC_HighSpeedTimingNone*** MMC card using none high-speed timing.
- kMMC_HighSpeedTiming*** MMC card using high-speed timing.
- kMMC_HighSpeed200Timing*** MMC card high speed 200 timing.
- kMMC_HighSpeed400Timing*** MMC card high speed 400 timing.
- kMMC_EnhanceHighSpeed400Timing*** MMC card high speed 400 timing.

46.7.4.52 enum mmc_data_bus_width_t

Enumerator

- kMMC_DataBusWidth1bit*** MMC data bus width is 1 bit.
- kMMC_DataBusWidth4bit*** MMC data bus width is 4 bits.
- kMMC_DataBusWidth8bit*** MMC data bus width is 8 bits.
- kMMC_DataBusWidth4bitDDR*** MMC data bus width is 4 bits ddr.
- kMMC_DataBusWidth8bitDDR*** MMC data bus width is 8 bits ddr.
- kMMC_DataBusWidth8bitDDRSTROBE*** MMC data bus width is 8 bits ddr strobe mode.

46.7.4.53 enum mmc_boot_partition_enable_t

Enumerator

kMMC_BootPartitionEnableNot Device not boot enabled (default)

kMMC_BootPartitionEnablePartition1 Boot partition 1 enabled for boot.

kMMC_BootPartitionEnablePartition2 Boot partition 2 enabled for boot.

kMMC_BootPartitionEnableUserAera User area enabled for boot.

46.7.4.54 enum mmc_boot_timing_mode_t

Enumerator

kMMC_BootModeSDRWithDefaultTiming boot mode single data rate with backward compatible timings

kMMC_BootModeSDRWithHighSpeedTiming boot mode single data rate with high speed timing

kMMC_BootModeDDRTiming boot mode dual date rate

46.7.4.55 enum mmc_boot_partition_wp_t

Enumerator

kMMC_BootPartitionWPDisable boot partition write protection disable

kMMC_BootPartitionPwrWPToBothPartition power on period write protection apply to both boot partitions

kMMC_BootPartitionPermWPToBothPartition permanent write protection apply to both boot partitions

kMMC_BootPartitionPwrWPToPartition1 power on period write protection apply to partition1

kMMC_BootPartitionPwrWPToPartition2 power on period write protection apply to partition2

kMMC_BootPartitionPermWPToPartition1 permanent write protection apply to partition1

kMMC_BootPartitionPermWPToPartition2 permanent write protection apply to partition2

kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2 permanent write protection apply to partition1, power on period write protection apply to partition2

kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 permanent write protection apply to partition2, power on period write protection apply to partition1

46.7.4.56 anonymous enum

Enumerator

kMMC_BootPartitionNotProtected boot partition not protected

kMMC_BootPartitionPwrProtected boot partition is power on period write protected

kMMC_BootPartitionPermProtected boot partition is permanently protected

46.7.4.57 enum mmc_access_partition_t

Enumerator

kMMC_AccessPartitionUserAera No access to boot partition (default), normal partition.
kMMC_AccessPartitionBoot1 Read/Write boot partition 1.
kMMC_AccessPartitionBoot2 Read/Write boot partition 2.
kMMC_AccessRPMB Replay protected mem block.
kMMC_AccessGeneralPurposePartition1 access to general purpose partition 1
kMMC_AccessGeneralPurposePartition2 access to general purpose partition 2
kMMC_AccessGeneralPurposePartition3 access to general purpose partition 3
kMMC_AccessGeneralPurposePartition4 access to general purpose partition 4

46.7.4.58 anonymous enum

Enumerator

kMMC_CsdReadBlockPartialFlag Partial blocks for read allowed.
kMMC_CsdWriteBlockMisalignFlag Write block misalignment.
kMMC_CsdReadBlockMisalignFlag Read block misalignment.
kMMC_CsdDsrImplementedFlag DSR implemented.
kMMC_CsdWriteProtectGroupEnabledFlag Write protect group enabled.
kMMC_CsdWriteBlockPartialFlag Partial blocks for write allowed.
kMMC_ContentProtectApplicationFlag Content protect application.
kMMC_CsdFileFormatGroupFlag File format group.
kMMC_CsdCopyFlag Copy flag.
kMMC_CsdPermanentWriteProtectFlag Permanent write protection.
kMMC_CsdTemporaryWriteProtectFlag Temporary write protection.

46.7.4.59 enum mmc_extended_csd_access_mode_t

Enumerator

kMMC_ExtendedCsdAccessModeCommandSet Command set related setting.
kMMC_ExtendedCsdAccessModeSetBits Set bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeClearBits Clear bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeWriteBits Write a value to specific byte in Extended CSD.

46.7.4.60 enum mmc_extended_csd_index_t

Enumerator

kMMC_ExtendedCsdIndexFlushCache flush cache
kMMC_ExtendedCsdIndexCacheControl cache control

kMMC_ExtendedCsdIndexBootPartitionWP Boot partition write protect.
kMMC_ExtendedCsdIndexEraseGroupDefinition Erase Group Def.
kMMC_ExtendedCsdIndexBootBusConditions Boot Bus conditions.
kMMC_ExtendedCsdIndexBootConfigWP Boot config write protect.
kMMC_ExtendedCsdIndexPartitionConfig Partition Config, before BOOT_CONFIG.
kMMC_ExtendedCsdIndexBusWidth Bus Width.
kMMC_ExtendedCsdIndexHighSpeedTiming High-speed Timing.
kMMC_ExtendedCsdIndexPowerClass Power Class.
kMMC_ExtendedCsdIndexCommandSet Command Set.

46.7.4.61 anonymous enum

Enumerator

kMMC_DriverStrength0 Driver type0 ,nominal impedance 50ohm.
kMMC_DriverStrength1 Driver type1 ,nominal impedance 33ohm.
kMMC_DriverStrength2 Driver type2 ,nominal impedance 66ohm.
kMMC_DriverStrength3 Driver type3 ,nominal impedance 100ohm.
kMMC_DriverStrength4 Driver type4 ,nominal impedance 40ohm.

46.7.4.62 enum mmc_extended_csd_flags_t

Enumerator

kMMC_ExtCsdExtPartitionSupport partitioning support[160]
kMMC_ExtCsdEnhancePartitionSupport partitioning support[160]
kMMC_ExtCsdPartitioningSupport partitioning support[160]
kMMC_ExtCsdPrgCIDCSDInDDRModeSupport CMD26 and CMD27 are support dual data rate [130].
kMMC_ExtCsdBKOpsSupport background operation feature support [502]
kMMC_ExtCsdDataTagSupport data tag support[499]
kMMC_ExtCsdModeOperationCodeSupport mode operation code support[493]

46.7.4.63 enum mmc_boot_mode_t

Enumerator

kMMC_BootModeNormal Normal boot.
kMMC_BootModeAlternative Alternative boot.

46.7.5 Function Documentation

46.7.5.1 **status_t SDMMC_SelectCard (*sdmmchost_t * host*, *uint32_t relativeAddress*,
bool isSelected)**

Parameters

<i>host</i>	host handler.
<i>relativeAddress</i>	Relative address.
<i>isSelected</i>	True to put card into transfer state.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

46.7.5.2 status_t SDMMC_SendApplicationCommand (*sdmmchost_t * host, uint32_t relativeAddress*)

Parameters

<i>host</i>	host handler.
<i>relativeAddress</i>	Card relative address.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_CardNotSupport</i>	Card doesn't support.
<i>kStatus_Success</i>	Operate successfully.

46.7.5.3 status_t SDMMC_SetBlockCount (*sdmmchost_t * host, uint32_t blockCount*)

Parameters

<i>host</i>	host handler.
<i>blockCount</i>	Block count.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

46.7.5.4 status_t SDMMC_GoIdle (*sdmmchost_t * host*)

Parameters

<i>host</i>	host handler.
-------------	---------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

46.7.5.5 status_t SDMMC_SetBlockSize (*sdmmchost_t * host, uint32_t blockSize*)

Parameters

<i>host</i>	host handler.
<i>blockSize</i>	Block size.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

46.7.5.6 status_t SDMMC_SetCardInactive (*sdmmchost_t * host*)

Parameters

<i>host</i>	host handler.
-------------	---------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

Chapter 47

CODEC Driver

47.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- [CODEC Common Driver](#)
- [CODEC I2C Driver](#)
- [Wm8962](#)

47.2 CODEC Common Driver

47.2.1 Overview

The codec common driver provides a codec control abstraction interface.

Modules

- [WM8962 Adapter](#)

Data Structures

- struct `codec_config_t`
Initialize structure of the codec. [More...](#)
- struct `codec_capability_t`
codec capability [More...](#)
- struct `codec_handle_t`
Codec handle definition. [More...](#)

Macros

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

Enumerations

- enum {

 `kStatus_CODEC_NotSupport` = MAKE_STATUS(kStatusGroup_CODEC, 0U),

 `kStatus_CODEC_DeviceNotRegistered` = MAKE_STATUS(kStatusGroup_CODEC, 1U),

 `kStatus_CODEC_I2CBusInitialFailed`,

 `kStatus_CODEC_I2CCommandTransferFailed` }

CODEC status.
- enum `codec_audio_protocol_t` {

 `kCODEC_BusI2S` = 0U,

 `kCODEC_BusLeftJustified` = 1U,

 `kCODEC_BusRightJustified` = 2U,

 `kCODEC_BusPCMA` = 3U,

 `kCODEC_BusPCMB` = 4U,

 `kCODEC_BusTDM` = 5U }

AUDIO format definition.
- enum {

```

kCODEC_AudioSampleRate8KHz = 8000U,
kCODEC_AudioSampleRate11025Hz = 11025U,
kCODEC_AudioSampleRate12KHz = 12000U,
kCODEC_AudioSampleRate16KHz = 16000U,
kCODEC_AudioSampleRate22050Hz = 22050U,
kCODEC_AudioSampleRate24KHz = 24000U,
kCODEC_AudioSampleRate32KHz = 32000U,
kCODEC_AudioSampleRate44100Hz = 44100U,
kCODEC_AudioSampleRate48KHz = 48000U,
kCODEC_AudioSampleRate96KHz = 96000U,
kCODEC_AudioSampleRate192KHz = 192000U,
kCODEC_AudioSampleRate384KHz = 384000U }

    audio sample rate definition
• enum {
    kCODEC_AudioBitWidth16bit = 16U,
    kCODEC_AudioBitWidth20bit = 20U,
    kCODEC_AudioBitWidth24bit = 24U,
    kCODEC_AudioBitWidth32bit = 32U }

    audio bit width
• enum codec_module_t {
    kCODEC_ModuleADC = 0U,
    kCODEC_ModuleDAC = 1U,
    kCODEC_ModulePGA = 2U,
    kCODEC_ModuleHeadphone = 3U,
    kCODEC_ModuleSpeaker = 4U,
    kCODEC_ModuleLinein = 5U,
    kCODEC_ModuleLineout = 6U,
    kCODEC_ModuleVref = 7U,
    kCODEC_ModuleMicbias = 8U,
    kCODEC_ModuleMic = 9U,
    kCODEC_ModuleI2SIn = 10U,
    kCODEC_ModuleI2SOut = 11U,
    kCODEC_ModuleMixer = 12U }

    audio codec module
• enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }

    audio codec module control cmd
• enum {
    kCODEC_ModuleI2SInInterfacePCM = 0U,
    kCODEC_ModuleI2SInInterfaceDSD = 1U }

    audio codec module digital interface
• enum {
    kCODEC_RecordSourceDifferentialLine = 1U,
    kCODEC_RecordSourceLineInput = 2U,
    kCODEC_RecordSourceDifferentialMic = 4U,
    kCODEC_RecordSourceDigitalMic = 8U,
    kCODEC_RecordSourceSingleEndMic = 16U }

```

- audio codec module record source value*
- enum {

kCODEC_RecordChannelLeft1 = 1U,
 kCODEC_RecordChannelLeft2 = 2U,
 kCODEC_RecordChannelLeft3 = 4U,
 kCODEC_RecordChannelRight1 = 1U,
 kCODEC_RecordChannelRight2 = 2U,
 kCODEC_RecordChannelRight3 = 4U,
 kCODEC_RecordChannelDifferentialPositive1 = 1U,
 kCODEC_RecordChannelDifferentialPositive2 = 2U,
 kCODEC_RecordChannelDifferentialPositive3 = 4U,
 kCODEC_RecordChannelDifferentialNegative1 = 8U,
 kCODEC_RecordChannelDifferentialNegative2 = 16U,
 kCODEC_RecordChannelDifferentialNegative3 = 32U }
- audio codec record channel*
- enum {

kCODEC_PlaySourcePGA = 1U,
 kCODEC_PlaySourceInput = 2U,
 kCODEC_PlaySourceDAC = 4U,
 kCODEC_PlaySourceMixerIn = 1U,
 kCODEC_PlaySourceMixerInLeft = 2U,
 kCODEC_PlaySourceMixerInRight = 4U,
 kCODEC_PlaySourceAux = 8U }
- audio codec module play source value*
- enum {

kCODEC_PlayChannelHeadphoneLeft = 1U,
 kCODEC_PlayChannelHeadphoneRight = 2U,
 kCODEC_PlayChannelSpeakerLeft = 4U,
 kCODEC_PlayChannelSpeakerRight = 8U,
 kCODEC_PlayChannelLineOutLeft = 16U,
 kCODEC_PlayChannelLineOutRight = 32U,
 kCODEC_PlayChannelLeft0 = 1U,
 kCODEC_PlayChannelRight0 = 2U,
 kCODEC_PlayChannelLeft1 = 4U,
 kCODEC_PlayChannelRight1 = 8U,
 kCODEC_PlayChannelLeft2 = 16U,
 kCODEC_PlayChannelRight2 = 32U,
 kCODEC_PlayChannelLeft3 = 64U,
 kCODEC_PlayChannelRight3 = 128U }
- codec play channel*
- enum {

```
kCODEC_VolumeHeadphoneLeft = 1U,  
kCODEC_VolumeHeadphoneRight = 2U,  
kCODEC_VolumeSpeakerLeft = 4U,  
kCODEC_VolumeSpeakerRight = 8U,  
kCODEC_VolumeLineOutLeft = 16U,  
kCODEC_VolumeLineOutRight = 32U,  
kCODEC_VolumeLeft0 = 1UL << 0U,  
kCODEC_VolumeRight0 = 1UL << 1U,  
kCODEC_VolumeLeft1 = 1UL << 2U,  
kCODEC_VolumeRight1 = 1UL << 3U,  
kCODEC_VolumeLeft2 = 1UL << 4U,  
kCODEC_VolumeRight2 = 1UL << 5U,  
kCODEC_VolumeLeft3 = 1UL << 6U,  
kCODEC_VolumeRight3 = 1UL << 7U,  
kCODEC_VolumeDAC = 1UL << 8U }
```

codec volume setting

- enum {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

audio codec capability

Functions

- `status_t CODEC_Init (codec_handle_t *handle, codec_config_t *config)`
Codec initialization.
- `status_t CODEC_Deinit (codec_handle_t *handle)`
Codec de-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.

47.2.2 Data Structure Documentation

47.2.2.1 struct codec_config_t

Data Fields

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

47.2.2.2 struct codec_capability_t

Data Fields

- `uint32_t codecModuleCapability`
codec module capability
- `uint32_t codecPlayCapability`
codec play capability
- `uint32_t codecRecordCapability`
codec record capability
- `uint32_t codecVolumeCapability`
codec volume capability

47.2.2.3 struct _codec_handle

codec handle declaration

- Application should allocate a buffer with CODEC_HANDLE_SIZE for handle definition, such as `uint8_t codecHandleBuffer[CODEC_HANDLE_SIZE]; codec_handle_t *codecHandle = codecHandleBuffer;`

Data Fields

- `codec_config_t * codecConfig`
codec configuration function pointer
- `const codec_capability_t * codecCapability`
codec capability
- `uint8_t codecDevHandle [HAL_CODEC_HANDLER_SIZE]`
codec device handle

47.2.3 Macro Definition Documentation

47.2.3.1 #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))

47.2.4 Enumeration Type Documentation

47.2.4.1 anonymous enum

Enumerator

`kStatus_CODEC_NotSupport` CODEC not support status.

`kStatus_CODEC_DeviceNotRegistered` CODEC device register failed status.

`kStatus_CODEC_I2CBusInitialFailed` CODEC i2c bus initialization failed status.

`kStatus_CODEC_I2CCommandTransferFailed` CODEC i2c bus command transfer failed status.

47.2.4.2 enum codec_audio_protocol_t

Enumerator

- kCODEC_BusI2S* I2S type.
- kCODEC_BusLeftJustified* Left justified mode.
- kCODEC_BusRightJustified* Right justified mode.
- kCODEC_BusPCMA* DSP/PCM A mode.
- kCODEC_BusPCMB* DSP/PCM B mode.
- kCODEC_BusTDM* TDM mode.

47.2.4.3 anonymous enum

Enumerator

- kCODEC_AudioSampleRate8KHz* Sample rate 8000 Hz.
- kCODEC_AudioSampleRate11025Hz* Sample rate 11025 Hz.
- kCODEC_AudioSampleRate12KHz* Sample rate 12000 Hz.
- kCODEC_AudioSampleRate16KHz* Sample rate 16000 Hz.
- kCODEC_AudioSampleRate22050Hz* Sample rate 22050 Hz.
- kCODEC_AudioSampleRate24KHz* Sample rate 24000 Hz.
- kCODEC_AudioSampleRate32KHz* Sample rate 32000 Hz.
- kCODEC_AudioSampleRate44100Hz* Sample rate 44100 Hz.
- kCODEC_AudioSampleRate48KHz* Sample rate 48000 Hz.
- kCODEC_AudioSampleRate96KHz* Sample rate 96000 Hz.
- kCODEC_AudioSampleRate192KHz* Sample rate 192000 Hz.
- kCODEC_AudioSampleRate384KHz* Sample rate 384000 Hz.

47.2.4.4 anonymous enum

Enumerator

- kCODEC_AudioBitWidth16bit* audio bit width 16
- kCODEC_AudioBitWidth20bit* audio bit width 20
- kCODEC_AudioBitWidth24bit* audio bit width 24
- kCODEC_AudioBitWidth32bit* audio bit width 32

47.2.4.5 enum codec_module_t

Enumerator

- kCODEC_ModuleADC* codec module ADC
- kCODEC_ModuleDAC* codec module DAC
- kCODEC_ModulePGA* codec module PGA
- kCODEC_ModuleHeadphone* codec module headphone

kCODEC_ModuleSpeaker codec module speaker
kCODEC_ModuleLinein codec module linein
kCODEC_ModuleLineout codec module lineout
kCODEC_ModuleVref codec module VREF
kCODEC_ModuleMicbias codec module MIC BIAS
kCODEC_ModuleMic codec module MIC
kCODEC_ModuleI2SIn codec module I2S in
kCODEC_ModuleI2SOut codec module I2S out
kCODEC_ModuleMixer codec module mixer

47.2.4.6 enum codec_module_ctrl_cmd_t

Enumerator

kCODEC_ModuleSwitchI2SInInterface module digital interface siwtch.

47.2.4.7 anonymous enum

Enumerator

kCODEC_ModuleI2SInInterfacePCM Pcm interface.
kCODEC_ModuleI2SInInterfaceDSD DSD interface.

47.2.4.8 anonymous enum

Enumerator

kCODEC_RecordSourceDifferentialLine record source from differential line
kCODEC_RecordSourceLineInput record source from line input
kCODEC_RecordSourceDifferentialMic record source from differential mic
kCODEC_RecordSourceDigitalMic record source from digital microphone
kCODEC_RecordSourceSingleEndMic record source from single microphone

47.2.4.9 anonymous enum

Enumerator

kCODEC_RecordChannelLeft1 left record channel 1
kCODEC_RecordChannelLeft2 left record channel 2
kCODEC_RecordChannelLeft3 left record channel 3
kCODEC_RecordChannelRight1 right record channel 1
kCODEC_RecordChannelRight2 right record channel 2
kCODEC_RecordChannelRight3 right record channel 3
kCODEC_RecordChannelDifferentialPositive1 differential positive record channel 1

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

47.2.4.10 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

47.2.4.11 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

47.2.4.12 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

47.2.4.13 anonymous enum

Enumerator

kCODEC_SupportModuleADC codec capability of module ADC
kCODEC_SupportModuleDAC codec capability of module DAC
kCODEC_SupportModulePGA codec capability of module PGA
kCODEC_SupportModuleHeadphone codec capability of module headphone
kCODEC_SupportModuleSpeaker codec capability of module speaker
kCODEC_SupportModuleLinein codec capability of module linein
kCODEC_SupportModuleLineout codec capability of module lineout
kCODEC_SupportModuleVref codec capability of module vref
kCODEC_SupportModuleMicbias codec capability of module mic bias
kCODEC_SupportModuleMic codec capability of module mic bias
kCODEC_SupportModuleI2SIn codec capability of module I2S in
kCODEC_SupportModuleI2SOut codec capability of module I2S out
kCODEC_SupportModuleMixer codec capability of module mixer
kCODEC_SupportModuleI2SInSwitchInterface codec capability of module I2S in switch interface

kCODEC_SupportPlayChannelLeft0 codec capability of play channel left 0
kCODEC_SupportPlayChannelRight0 codec capability of play channel right 0
kCODEC_SupportPlayChannelLeft1 codec capability of play channel left 1
kCODEC_SupportPlayChannelRight1 codec capability of play channel right 1
kCODEC_SupportPlayChannelLeft2 codec capability of play channel left 2
kCODEC_SupportPlayChannelRight2 codec capability of play channel right 2
kCODEC_SupportPlayChannelLeft3 codec capability of play channel left 3
kCODEC_SupportPlayChannelRight3 codec capability of play channel right 3
kCODEC_SupportPlaySourcePGA codec capability of set playback source PGA
kCODEC_SupportPlaySourceInput codec capability of set playback source INPUT
kCODEC_SupportPlaySourceDAC codec capability of set playback source DAC
kCODEC_SupportPlaySourceMixerIn codec capability of set play source Mixer in
kCODEC_SupportPlaySourceMixerInLeft codec capability of set play source Mixer in left
kCODEC_SupportPlaySourceMixerInRight codec capability of set play source Mixer in right

kCODEC_SupportPlaySourceAux codec capability of set play source aux

kCODEC_SupportRecordSourceDifferentialLine codec capability of record source differential line

kCODEC_SupportRecordSourceLineInput codec capability of record source line input

kCODEC_SupportRecordSourceDifferentialMic codec capability of record source differential mic

kCODEC_SupportRecordSourceDigitalMic codec capability of record digital mic

kCODEC_SupportRecordSourceSingleEndMic codec capability of single end mic

kCODEC_SupportRecordChannelLeft1 left record channel 1

kCODEC_SupportRecordChannelLeft2 left record channel 2

kCODEC_SupportRecordChannelLeft3 left record channel 3

kCODEC_SupportRecordChannelRight1 right record channel 1

kCODEC_SupportRecordChannelRight2 right record channel 2

kCODEC_SupportRecordChannelRight3 right record channel 3

47.2.5 Function Documentation

47.2.5.1 status_t CODEC_Init (***codec_handle_t * handle***, ***codec_config_t * config***)

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configurations.

Returns

kStatus_Success is success, else de-initial failed.

47.2.5.2 status_t CODEC_Deinit (***codec_handle_t * handle***)

Parameters

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

Returns

kStatus_Success is success, else de-initial failed.

47.2.5.3 status_t CODEC_SetFormat (***codec_handle_t * handle***, ***uint32_t mclk***, ***uint32_t sampleRate***, ***uint32_t bitWidth***)

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

47.2.5.4 status_t CODEC_ModuleControl (*codec_handle_t * handle*, *codec_module_ctrl_cmd_t cmd*, *uint32_t data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature.

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

47.2.5.5 status_t CODEC_SetVolume (*codec_handle_t * handle*, *uint32_t channel*, *uint32_t volume*)

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec volume channel, can be a value or combine value of _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.

47.2.5.6 status_t CODEC_SetMute (*codec_handle_t * handle*, *uint32_t channel*, *bool mute*)

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel.
<i>mute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

47.2.5.7 status_t CODEC_SetPower (*codec_handle_t * handle*, *codec_module_t module*, *bool powerOn*)

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

47.2.5.8 status_t CODEC_SetRecord (*codec_handle_t * handle*, *uint32_t recordSource*)

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

47.2.5.9 status_t CODEC_SetRecordChannel (*codec_handle_t * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _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.

47.2.5.10 status_t CODEC_SetPlay (*codec_handle_t * handle, uint32_t playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

47.3 CODEC I2C Driver

47.3.1 Overview

The codec common driver provides a codec control abstraction interface.

Data Structures

- struct `codec_i2c_config_t`
CODEC I2C configurations structure. [More...](#)

Macros

- #define `CODEC_I2C_MASTER_HANDLER_SIZE` HAL_I2C_MASTER_HANDLE_SIZE
codec i2c handler

Enumerations

- enum `codec_reg_addr_t` {

`kCODEC_RegAddr8Bit` = 1U,
`kCODEC_RegAddr16Bit` = 2U }
CODEC device register address type.
- enum `codec_reg_width_t` {

`kCODEC_RegWidth8Bit` = 1U,
`kCODEC_RegWidth16Bit` = 2U,
`kCODEC_RegWidth32Bit` = 4U }
CODEC device register width.

Functions

- `status_t CODEC_I2C_Init` (void *handle, uint32_t i2cInstance, uint32_t i2cBaudrate, uint32_t i2cSourceClockHz)
Codec i2c bus initialization.
- `status_t CODEC_I2C_Deinit` (void *handle)
Codec i2c de-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.

47.3.2 Data Structure Documentation

47.3.2.1 struct codec_i2c_config_t

Data Fields

- `uint32_t codecI2CInstance`
i2c bus instance
- `uint32_t codecI2CSourceClock`
i2c bus source clock frequency

47.3.3 Enumeration Type Documentation

47.3.3.1 enum codec_reg_addr_t

Enumerator

`kCODEC_RegAddr8Bit` 8-bit register address.
`kCODEC_RegAddr16Bit` 16-bit register address.

47.3.3.2 enum codec_reg_width_t

Enumerator

`kCODEC_RegWidth8Bit` 8-bit register width.
`kCODEC_RegWidth16Bit` 16-bit register width.
`kCODEC_RegWidth32Bit` 32-bit register width.

47.3.4 Function Documentation

47.3.4.1 status_t CODEC_I2C_Init (void * handle, uint32_t i2cInstance, uint32_t i2cBaudrate, uint32_t i2cSourceClockHz)

Parameters

<code>handle</code>	i2c master handle.
<code>i2cInstance</code>	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.

47.3.4.2 status_t CODEC_I2C_Deinit (void * *handle*)

Parameters

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

Returns

kStatus_HAL_I2cSuccess is success, else deinitial failed.

47.3.4.3 status_t CODEC_I2C_Send (void * *handle*, uint8_t *deviceAddress*, uint32_t *subAddress*, uint8_t *subaddressSize*, uint8_t * *txBuff*, uint8_t *txBuffSize*)

Parameters

<i>handle</i>	i2c master handle.
<i>deviceAddress</i>	codec device address.
<i>subAddress</i>	register address.
<i>subaddressSize</i>	register address width.
<i>txBuff</i>	tx buffer pointer.
<i>txBuffSize</i>	tx buffer size.

Returns

kStatus_HAL_I2cSuccess is success, else send failed.

47.3.4.4 status_t CODEC_I2C_Receive (void * *handle*, uint8_t *deviceAddress*, uint32_t *subAddress*, uint8_t *subaddressSize*, uint8_t * *rxBuff*, uint8_t *rxBuffSize*)

Parameters

<i>handle</i>	i2c master handle.
<i>deviceAddress</i>	codec device address.
<i>subAddress</i>	register address.
<i>subaddressSize</i>	register address width.
<i>rxBuff</i>	rx buffer pointer.
<i>rxBuffSize</i>	rx buffer size.

Returns

kStatus_HAL_I2cSuccess is success, else receive failed.

47.3.5 WM8962 Adapter

47.3.5.1 Overview

The wm8962 adapter provides a codec unify control interface.

Macros

- #define `HAL_CODEC_WM8962_HANDLER_SIZE` (`WM8962_I2C_HANDLER_SIZE + 4`)
codec handler size

Functions

- `status_t HAL_CODEC_WM8962_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_WM8962_Deinit` (void *handle)
Codec de-initialization.
- `status_t HAL_CODEC_WM8962_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- `status_t HAL_CODEC_WM8962_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_WM8962_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_WM8962_SetPower` (void *handle, uint32_t module, bool powerOn)
set audio codec module power.
- `status_t HAL_CODEC_WM8962_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- `status_t HAL_CODEC_WM8962_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- `status_t HAL_CODEC_WM8962_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- `status_t HAL_CODEC_WM8962_ModuleControl` (void *handle, uint32_t cmd, uint32_t moduleData)
codec module control.
- static `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initialization.
- static `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initialization.
- static `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- static `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- static `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.

- static `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)
set audio codec module power.
- static `status_t HAL_CODEC_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- static `status_t HAL_CODEC_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- static `status_t HAL_CODEC_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- static `status_t HAL_CODEC_ModuleControl` (void *handle, uint32_t cmd, uint32_t moduleData)
codec module control.

47.3.5.2 Function Documentation

47.3.5.2.1 `status_t HAL_CODEC_WM8962_Init (void * handle, void * config)`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

`kStatus_Success` is success, else initial failed.

47.3.5.2.2 `status_t HAL_CODEC_WM8962_Deinit (void * handle)`

Parameters

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

Returns

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

47.3.5.2.3 `status_t HAL_CODEC_WM8962_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.

47.3.5.2.4 status_t HAL_CODEC_WM8962_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.

47.3.5.2.5 status_t HAL_CODEC_WM8962_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.

47.3.5.2.6 status_t HAL_CODEC_WM8962_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.

47.3.5.2.7 status_t HAL_CODEC_WM8962_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.

47.3.5.2.8 status_t HAL_CODEC_WM8962_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.

47.3.5.2.9 status_t HAL_CODEC_WM8962_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.

47.3.5.2.10 status_t HAL_CODEC_WM8962_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *moduleData*)

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>moduleData</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.

47.3.5.2.11 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.

47.3.5.2.12 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.

47.3.5.2.13 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.

47.3.5.2.14 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.

47.3.5.2.15 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.

47.3.5.2.16 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.

47.3.5.2.17 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.

47.3.5.2.18 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.

47.3.5.2.19 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.

47.3.5.2.20 static status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *moduleData*) [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>moduleData</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 48

Serial Manager

48.1 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 USB](#)
- [Serial Port Uart](#)

Data Structures

- struct [serial_manager_config_t](#)
serial manager config structure [More...](#)
- struct [serial_manager_callback_message_t](#)
Callback message structure. [More...](#)

Macros

- #define [SERIAL_MANAGER_NON_BLOCKING_MODE](#) (1U)
Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)
- #define [SERIAL_MANAGER_RING_BUFFER_FLOWCONTROL](#) (0U)
Enable or ring buffer flow control (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_UART](#) (0U)
Enable or disable uart port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_UART_DMA](#) (0U)
Enable or disable uart dma port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_USBCDC](#) (0U)
Enable or disable USB CDC port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SWO](#) (0U)
Enable or disable SWO port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_VIRTUAL](#) (0U)
Enable or disable USB CDC virtual port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_RPMSG](#) (0U)
Enable or disable rpmsg port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SPI_MASTER](#) (0U)
Enable or disable SPI Master port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_SPI_SLAVE](#) (0U)
Enable or disable SPI Slave port (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_BLE_WU](#) (0U)
Enable or disable BLE WU port (1 - enable, 0 - disable)

- #define **SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE** (1U)
Set the default delay time in ms used by SerialManager_WriteTimeDelay().
- #define **SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE** (1U)
Set the default delay time in ms used by SerialManager_ReadTimeDelay().
- #define **SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY** (0U)
Enable or disable SerialManager_Task() handle RX data available notify.
- #define **SERIAL_MANAGER_WRITE_HANDLE_SIZE** (44U)
Set serial manager write handle size.
- #define **SERIAL_MANAGER_USE_COMMON_TASK** (0U)
SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.
- #define **SERIAL_MANAGER_HANDLE_SIZE** (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 124U)
Definition of serial manager handle size.
- #define **SERIAL_MANAGER_HANDLE_DEFINE**(name) uint32_t name[((**SERIAL_MANAGER_HANDLE_SIZE** + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager handle.
- #define **SERIAL_MANAGER_WRITE_HANDLE_DEFINE**(name) uint32_t name[((**SERIAL_MANAGER_WRITE_HANDLE_SIZE** + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager write handle.
- #define **SERIAL_MANAGER_READ_HANDLE_DEFINE**(name) uint32_t name[((**SERIAL_MANAGER_READ_HANDLE_SIZE** + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager read handle.
- #define **SERIAL_MANAGER_TASK_PRIORITY** (2U)
Macro to set serial manager task priority.
- #define **SERIAL_MANAGER_TASK_STACK_SIZE** (1000U)
Macro to set serial manager task stack size.

Typedefs

- typedef void * **serial_handle_t**
The handle of the serial manager module.
- typedef void * **serial_write_handle_t**
The write handle of the serial manager module.
- typedef void * **serial_read_handle_t**
The read handle of the serial manager module.
- typedef void(* **serial_manager_callback_t**)(void *callbackParam, **serial_manager_callback_message_t** *message, **serial_manager_status_t** status)
serial manager callback function
- typedef int32_t(* **serial_manager_lowpower_critical_callback_t**)(int32_t power_mode)
serial manager Lowpower Critical callback function

Enumerations

- enum `serial_port_type_t` {

 `kSerialPort_None` = 0U,

 `kSerialPort_Uart` = 1U,

 `kSerialPort_UsbCdc`,

 `kSerialPort_Swo`,

 `kSerialPort_Virtual`,

 `kSerialPort_Rpmsg`,

 `kSerialPort_UartDma`,

 `kSerialPort_SpiMaster`,

 `kSerialPort_SpiSlave`,

 `kSerialPort_BleWu` }

 serial port type
- enum `serial_manager_type_t` {

 `kSerialManager_NonBlocking` = 0x0U,

 `kSerialManager_Blocking` = 0x8F41U }

 serial manager type
- enum `serial_manager_status_t` {

 `kStatus_SerialManager_Success` = `kStatus_Success`,

 `kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,

 `kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,

 `kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,

 `kStatus_SerialManager_Canceled`,

 `kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,

 `kStatus_SerialManager_RingBufferOverflow`,

 `kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }

 serial manager error code

Functions

- `serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const serial_manager_config_t *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 *pfnCallback)`
This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.

48.2 Data Structure Documentation

48.2.1 struct serial_manager_config_t

Data Fields

- `uint8_t * ringBuffer`
Ring buffer address, it is used to buffer data received by the hardware.
- `uint32_t ringBufferSize`
The size of the ring buffer.
- `serial_port_type_t type`
Serial port type.
- `serial_manager_type_t blockType`

- *Serial manager port type.*
- `void * portConfig`
Serial port configuration.

Field Documentation

(1) `uint8_t* serial_manager_config_t::ringBuffer`

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

48.2.2 `struct serial_manager_callback_message_t`

Data Fields

- `uint8_t * buffer`
Transferred buffer.
- `uint32_t length`
Transferred data length.

48.3 Macro Definition Documentation

48.3.1 `#define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)`

48.3.2 `#define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)`

48.3.3 `#define SERIAL_MANAGER_USE_COMMON_TASK (0U)`

Macro to determine whether use common task.

48.3.4 `#define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 124U)`

48.3.5 `#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.
-------------	---

48.3.6 #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.
-------------	---

48.3.7 #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.
-------------	--

48.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

48.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

48.4 Enumeration Type Documentation

48.4.1 enum serial_port_type_t

Enumerator

- kSerialPort_None* Serial port is none.
- kSerialPort_Uart* Serial port UART.
- kSerialPort_UsbCdc* Serial port USB CDC.
- kSerialPort_Swo* Serial port SWO.
- kSerialPort_Virtual* Serial port Virtual.
- kSerialPort_Rpmsg* Serial port RPMSG.
- kSerialPort_UartDma* Serial port UART DMA.
- kSerialPort_SpiMaster* Serial port SPIMASTER.
- kSerialPort_SpiSlave* Serial port SPISLAVE.
- kSerialPort_BleWu* Serial port BLE WU.

48.4.2 enum serial_manager_type_t

Enumerator

- kSerialManager_NonBlocking* None blocking handle.
- kSerialManager_Blocking* Blocking handle.

48.4.3 enum serial_manager_status_t

Enumerator

- kStatus_SerialManager_Success* Success.
- kStatus_SerialManager_Error* Failed.
- kStatus_SerialManager_Busy* Busy.
- kStatus_SerialManager_Notify* Ring buffer is not empty.
- kStatus_SerialManager_Canceled* the non-blocking request is canceled

kStatus_SerialManager_HandleConflict The handle is opened.

kStatus_SerialManager_RingBufferOverflow The ring buffer is overflowed.

kStatus_SerialManager_NotConnected The host is not connected.

48.5 Function Documentation

48.5.1 `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 <code>SERIAL_MANAGER_HANDLE_SIZE</code> 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: <code>SERIAL_MANAGER_HANDLE_DEFINE(serialHandle)</code> ; 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

<code>kStatus_SerialManager_Error</code>	An error occurred.
<code>kStatus_SerialManager_Success</code>	The Serial Manager module initialization succeed.

48.5.2 `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

<code>kStatus_SerialManager_Success</code>	The serial manager de-initialization succeed.
<code>kStatus_SerialManager_Busy</code>	Opened reading or writing handle is not closed.

48.5.3 `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: SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle) ; 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);
*
*
```

48.5.4 **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.
---------------------------------------	-------------------------------

48.5.5 **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: SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle) ; or uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

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));
*
```

48.5.6 **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.
---------------------------------------	-------------------------------

48.5.7 **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.

48.5.8 `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_- Busy</code>	Previous transmission still not finished; data not all received yet.
<code>kStatus_SerialManager_- Error</code>	An error occurred.

48.5.9 `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_- Success</code>	Successfully sent all data.
<code>kStatus_SerialManager_- Busy</code>	Previous transmission still not finished; data not all sent yet.
<code>kStatus_SerialManager_- Error</code>	An error occurred.

48.5.10 `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.

48.5.11 `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.

48.5.12 `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.

48.5.13 `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.

48.5.14 **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.
---------------------------------------	------------------------------------

48.5.15 **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.
---------------------------------------	------------------------------------

48.5.16 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.
-------------	------------------

48.5.17 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.
---------------------------------------	-----------------------

48.5.18 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.
---------------------------------------	-----------------------

48.5.19 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.
-------------------	---

48.6 Serial Port Uart

48.6.1 Overview

Macros

- #define **SERIAL_PORT_UART_DMA_RECEIVE_DATA_LENGTH** (64U)
serial port uart handle size
- #define **SERIAL_USE_CONFIGURE_STRUCTURE** (0U)
Enable or disable the configure structure pointer.

Enumerations

- enum **serial_port_uart_parity_mode_t** {

 kSerialManager_UartParityDisabled = 0x0U,

 kSerialManager_UartParityEven = 0x2U,

 kSerialManager_UartParityOdd = 0x3U }

serial port uart parity mode
- enum **serial_port_uart_stop_bit_count_t** {

 kSerialManager_UartOneStopBit = 0U,

 kSerialManager_UartTwoStopBit = 1U }

serial port uart stop bit count

48.6.2 Enumeration Type Documentation

48.6.2.1 enum serial_port_uart_parity_mode_t

Enumerator

- kSerialManager_UartParityDisabled*** Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

48.6.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

- kSerialManager_UartOneStopBit*** One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

48.7 Serial Port USB

48.7.1 Overview

Modules

- [USB Device Configuration](#)

Data Structures

- struct [serial_port_usb_cdc_config_t](#)
serial port usb config struct [More...](#)

Macros

- #define [SERIAL_PORT_USB_CDC_HANDLE_SIZE](#) (72U)
serial port usb handle size
- #define [USB_DEVICE_INTERRUPT_PRIORITY](#) (3U)
USB interrupt priority.

Enumerations

- enum [serial_port_usb_cdc_controller_index_t](#) {
 kSerialManager_UsbControllerKhci0 = 0U,
 kSerialManager_UsbControllerKhci1 = 1U,
 kSerialManager_UsbControllerEhci0 = 2U,
 kSerialManager_UsbControllerEhci1 = 3U,
 kSerialManager_UsbControllerLpcIp3511Fs0 = 4U,
 kSerialManager_UsbControllerLpcIp3511Fs1 = 5U,
 kSerialManager_UsbControllerLpcIp3511Hs0 = 6U,
 kSerialManager_UsbControllerLpcIp3511Hs1 = 7U,
 kSerialManager_UsbControllerOhci0 = 8U,
 kSerialManager_UsbControllerOhci1 = 9U,
 kSerialManager_UsbControllerIp3516Hs0 = 10U,
 kSerialManager_UsbControllerIp3516Hs1 = 11U }
USB controller ID.

48.7.2 Data Structure Documentation

48.7.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- `serial_port_usb_cdc_controller_index_t controllerIndex`
controller index

48.7.3 Enumeration Type Documentation

48.7.3.1 enum serial_port_usb_cdc_controller_index_t

Enumerator

`kSerialManager_UsbControllerKhci0` KHCI 0U.

`kSerialManager_UsbControllerKhci1` KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerEhci0` EHCI 0U.

`kSerialManager_UsbControllerEhci1` EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerLpcIp3511Fs0` LPC USB IP3511 FS controller 0.

`kSerialManager_UsbControllerLpcIp3511Fs1` LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerLpcIp3511Hs0` LPC USB IP3511 HS controller 0.

`kSerialManager_UsbControllerLpcIp3511Hs1` LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerOhci0` OHCI 0U.

`kSerialManager_UsbControllerOhci1` OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerIp3516Hs0` IP3516HS 0U.

`kSerialManager_UsbControllerIp3516Hs1` IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

48.7.4 USB Device Configuration

48.8 Serial Port SWO

48.8.1 Overview

Data Structures

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

Macros

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

Enumerations

- enum `serial_port_swo_protocol_t` {

`kSerialManager_SwoProtocolManchester` = 1U,
`kSerialManager_SwoProtocolNrz` = 2U }

serial port swo protocol

48.8.2 Data Structure Documentation

48.8.2.1 struct `serial_port_swo_config_t`

Data Fields

- `uint32_t clockRate`
clock rate
- `uint32_t baudRate`
baud rate
- `uint32_t port`
Port used to transfer data.
- `serial_port_swo_protocol_t protocol`
SWO protocol.

48.8.3 Enumeration Type Documentation

48.8.3.1 enum `serial_port_swo_protocol_t`

Enumerator

- `kSerialManager_SwoProtocolManchester` SWO Manchester protocol.
`kSerialManager_SwoProtocolNrz` SWO UART/NRZ protocol.

Chapter 49

Gpc

49.1 Overview

Enumerations

- enum { **kGPC_CM_DebugWakeupRequest** }
_gpc_cm_non_irq_wakeup_request GPC Non-IRQ wakeup request.
- enum {
 kGPC_Domain0 = 1UL << 0UL,
 kGPC_Domain1 = 1UL << 1UL,
 kGPC_Domain2 = 1UL << 2UL,
 kGPC_Domain3 = 1UL << 3UL,
 kGPC_Domain4 = 1UL << 4UL,
 kGPC_Domain5 = 1UL << 5UL,
 kGPC_Domain6 = 1UL << 6UL,
 kGPC_Domain7 = 1UL << 7UL,
 kGPC_Domain8 = 1UL << 8UL,
 kGPC_Domain9 = 1UL << 9UL,
 kGPC_Domain10 = 1UL << 10UL,
 kGPC_Domain11 = 1UL << 11UL,
 kGPC_Domain12 = 1UL << 12UL,
 kGPC_Domain13 = 1UL << 13UL,
 kGPC_Domain14 = 1UL << 14UL,
 kGPC_Domain15 = 1UL << 15UL }
- enum **gpc_cpu_slice_t** {
 kGPC_CPU0 = 0x0U,
 kGPC_CPU1 = 0x1U }
CPU slice.
- enum **gpc_cm_tran_step_t** {

```
kGPC_CM_SleepSsar = 0UL,
kGPC_CM_SleepLpcg = 1UL,
kGPC_CM_SleepPll = 2UL,
kGPC_CM_SleepIso = 3UL,
kGPC_CM_SleepReset = 4UL,
kGPC_CM_SleepPower = 5UL,
kGPC_CM_SleepSYS,
kGPC_CM_WakeupSYS,
kGPC_CM_WakeupPower = 8UL,
kGPC_CM_WakeupReset = 9UL,
kGPC_CM_WakeupIso = 10UL,
kGPC_CM_WakeupPll = 11UL,
kGPC_CM_WakeupLpcg = 12UL,
kGPC_CM_WakeupSsar = 13UL }
```

CPU mode transition step in sleep/wakeup sequence.

- enum `gpc_cpu_mode_t` {


```
kGPC_RunMode = 0x0UL,
kGPC_WaitMode = 0x1UL,
kGPC_StopMode = 0x2UL,
kGPC_SuspendMode = 0x3UL }
```

CPU mode.

- enum `gpc_ss_tran_step_t` {


```
kGPC_SS_Step0In = 0UL,
kGPC_SS_Step1In = 1UL,
kGPC_SS_Step2In = 2UL,
kGPC_SS_Step3In = 3UL,
kGPC_SS_DcdcIn = 4UL,
kGPC_SS_PmicIn = 5UL,
kGPC_SS_PmicOut = 6UL,
kGPC_SS_DcdcOut = 7UL,
kGPC_SS_Step3Out = 8UL,
kGPC_SS_Step2Out = 9UL,
kGPC_SS_Step1Out = 10UL,
kGPC_SS_Step0Out = 11UL }
```

GPC system sleep mode transition steps.

Driver version

- #define `FSL_GPC_RIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)

GPC driver version 2.1.0.

GPC global control

- void `GPC_AssignCpuDomain` (`gpc_cpu_slice_t` cpu, `uint32_t` domainID)

CPU mode control

- static void **GPC_CM_EnableCpuSleepHold** (`gpc_cpu_slice_t` slice, bool enable)
- static void **GPC_CM_SetNextCpuMode** (`gpc_cpu_slice_t` slice, `gpc_cpu_mode_t` mode)
Set the CPU mode on the next sleep event.
- static `gpc_cpu_mode_t` **GPC_CM_GetCurrentCpuMode** (`gpc_cpu_slice_t` slice)
Get current CPU mode.
- static `gpc_cpu_mode_t` **GPC_CM_GetPreviousCpuMode** (`gpc_cpu_slice_t` slice)
Get previous CPU mode.
- void **GPC_CM_EnableIrqWakeup** (`gpc_cpu_slice_t` slice, `uint32_t` irqId, bool enable)
Enable IRQ wakeup request.
- static void **GPC_CM_EnableNonIrqWakeup** (`gpc_cpu_slice_t` slice, `uint32_t` mask, bool enable)
Enable Non-IRQ wakeup request.
- bool **GPC_CM_GetIrqWakeupStatus** (`gpc_cpu_slice_t` slice, `uint32_t` irqId)
Get the status of the IRQ wakeup request.
- static bool **GPC_CM_GetNonIrqWakeupStatus** (`gpc_cpu_slice_t` slice, `uint32_t` mask)
Get the status of the Non-IRQ wakeup request.
- void **GPC_CM_EnableCpuModeTransitionStep** (`gpc_cpu_slice_t` slice, `gpc_cm_tran_step_t` step, bool enable)
brief Config the cpu mode transition step.
- void **GPC_CM_RequestSystemSleepMode** (`gpc_cpu_slice_t` slice, const `gpc_cpu_mode_t` mode)
Request the chip into system sleep mode.
- void **GPC_CM_ClearSystemSleepModeRequest** (`gpc_cpu_slice_t` slice, const `gpc_cpu_mode_t` mode)
Clear the system sleep mode request.
- static bool **GPC_CM_GetSystemSleepModeStatus** (`gpc_cpu_slice_t` slice, `uint32_t` mask)
Get the status of the CPU system sleep mode transition.

System sleep mode control

- void **GPC_SS_EnableSystemSleepTransitionStep** (`GPC_SYS_SLEEP_CTRL_Type` *base, `gpc_ss_tran_step_t` step, bool enable)
brief Config the system sleep transition step.
- static void **GPC_SS_SoftwareTriggerPMICStandby** (`GPC_SYS_SLEEP_CTRL_Type` *base, bool enable)
Trigger PMIC standby ON/OFF by software.
- static void **GPC_SS_SystemSleepTriggerPMICStandby** (`GPC_SYS_SLEEP_CTRL_Type` *base, bool enable)
brief Assert the PMIC standby request when system sleep.

49.2 Macro Definition Documentation

49.2.1 #define FSL_GPC_RIVER_VERSION (MAKE_VERSION(2, 1, 0))

49.3 Enumeration Type Documentation

49.3.1 anonymous enum

Enumerator

kGPC_CM_DebugWakeupRequest Debug wakeup request.

49.3.2 anonymous enum

Enumerator

- kGPC_Domain0* GPC domain 0.
- kGPC_Domain1* GPC domain 1.
- kGPC_Domain2* GPC domain 2.
- kGPC_Domain3* GPC domain 3.
- kGPC_Domain4* GPC domain 4.
- kGPC_Domain5* GPC domain 5.
- kGPC_Domain6* GPC domain 6.
- kGPC_Domain7* GPC domain 7.
- kGPC_Domain8* GPC domain 8.
- kGPC_Domain9* GPC domain 9.
- kGPC_Domain10* GPC domain 10.
- kGPC_Domain11* GPC domain 11.
- kGPC_Domain12* GPC domain 12.
- kGPC_Domain13* GPC domain 13.
- kGPC_Domain14* GPC domain 14.
- kGPC_Domain15* GPC domain 15.

49.3.3 enum gpc_cpu_slice_t

Enumerator

- kGPC_CPU0* CPU slice 0.
- kGPC_CPU1* CPU slice 1.

49.3.4 enum gpc_cm_tran_step_t

Enumerator

- kGPC_CM_SleepSsar* SSAR (State Save And Restore) sleep step.
- kGPC_CM_SleepLpcg* LPCG (Low Power Clock Gating) sleep step.
- kGPC_CM_SleepPll* PLL sleep step.
- kGPC_CM_SleepIso* ISO (Isolation) sleep step.
- kGPC_CM_SleepReset* Reset sleep step.
- kGPC_CM_SleepPower* Power sleep step.
- kGPC_CM_SleepSYS* System sleep sleep step. Note that this step is controlled by system sleep controller.
- kGPC_CM_WakeupSYS* System sleep wakeup step. Note that this step is controlled by system sleep controller.
- kGPC_CM_WakeupPower* Power wakeup step.

kGPC_CM_WakeupReset Reset wakeup step.
kGPC_CM_WakeupIso ISO wakeup step.
kGPC_CM_WakeupPll PLL wakeup step.
kGPC_CM_WakeupLpcg LPCG wakeup step.
kGPC_CM_WakeupSsar SSAR wakeup step.

49.3.5 enum gpc_cpu_mode_t

Enumerator

kGPC_RunMode Stay in RUN mode.
kGPC_WaitMode Transit to WAIT mode.
kGPC_StopMode Transit to STOP mode.
kGPC_SuspendMode Transit to SUSPEND mode.

49.3.6 enum gpc_ss_tran_step_t

Enumerator

kGPC_SS_Step0In Bias in step.
kGPC_SS_Step1In PLDO in step.
kGPC_SS_Step2In Bandgap in step.
kGPC_SS_Step3In LDO in step.
kGPC_SS_DcdcIn DCDC in step.
kGPC_SS_PmicIn PMIC in step.
kGPC_SS_PmicOut PMIC out step.
kGPC_SS_DcdcOut DCDC out step.
kGPC_SS_Step3Out LDO out step.
kGPC_SS_Step2Out Bandgap out step.
kGPC_SS_Step1Out PLDO out step.
kGPC_SS_Step0Out Bias out step.

49.4 Function Documentation

49.4.1 static void GPC_CM_SetNextCpuMode (gpc_cpu_slice_t slice, gpc_cpu_mode_t mode) [inline], [static]

This function configures the CPU mode that the CPU core will transmit to on next sleep event.

Note

This API must be called each time before entering sleep.

Parameters

<i>slice</i>	GPC CPU slice number.
<i>mode</i>	The CPU mode that the core will transmit to, refer to "gpc_cpu_mode_t".

49.4.2 static gpc_cpu_mode_t GPC_CM_GetCurrentCpuMode (gpc_cpu_slice_t *slice*) [inline], [static]

Parameters

<i>slice</i>	GPC CPU slice number.
--------------	-----------------------

Returns

The current CPU mode, in type of [gpc_cpu_mode_t](#).

49.4.3 static gpc_cpu_mode_t GPC_CM_GetPreviousCpuMode (gpc_cpu_slice_t *slice*) [inline], [static]

Parameters

<i>slice</i>	GPC CPU slice number.
--------------	-----------------------

Returns

The previous CPU mode, in type of [gpc_cpu_mode_t](#).

49.4.4 void GPC_CM_EnableIrqWakeup (gpc_cpu_slice_t *slice*, uint32_t *irqId*, bool *enable*)

This function enables the IRQ request which can wakeup the CPU platform.

Parameters

<i>slice</i>	GPC CPU slice number.
<i>irqId</i>	ID of the IRQ, accessible range is 0-255.
<i>enable</i>	Enable the IRQ request or not.

49.4.5 static void GPC_CM_EnableNonIrqWakeup (gpc_cpu_slice_t *slice*, uint32_t *mask*, bool *enable*) [inline], [static]

This function enables the non-IRQ request which can wakeup the CPU platform.

Parameters

<i>slice</i>	GPC CPU slice number.
<i>mask</i>	Non-IRQ type, refer to "_gpc_cm_non_irq_wakeup_request".
<i>enable</i>	Enable the Non-IRQ request or not.

49.4.6 bool GPC_CM_GetIrqWakeupStatus (gpc_cpu_slice_t *slice*, uint32_t *irqId*)

Parameters

<i>slice</i>	GPC CPU slice number.
<i>irqId</i>	ID of the IRQ, accessible range is 0-255.

Returns

Indicate the IRQ request is asserted or not.

49.4.7 static bool GPC_CM_GetNonIrqWakeupStatus (gpc_cpu_slice_t *slice*, uint32_t *mask*) [inline], [static]

Parameters

<i>slice</i>	GPC CPU slice number.
--------------	-----------------------

<i>mask</i>	Non-IRQ type, refer to "_gpc_cm_non_irq_wakeup_request".
-------------	--

Returns

Indicate the Non-IRQ request is asserted or not.

49.4.8 void GPC_CM_EnableCpuModeTransitionStep (*gpc_cpu_slice_t slice*, *gpc_cm_tran_step_t step*, *bool enable*)

param slice GPC CPU slice number. param step step type, refer to "gpc_cm_tran_step_t". param enable Used to control the transition step.

- **true** This step is enabled.
- **false** This step is disabled, GPC will skip this step and not send any request.

49.4.9 void GPC_CM_RequestSystemSleepMode (*gpc_cpu_slice_t slice*, *const gpc_cpu_mode_t mode*)

Parameters

<i>slice</i>	GPC CPU slice number.
<i>mode</i>	CPU mode. Refer to "gpc_cpu_mode_t".

49.4.10 void GPC_CM_ClearSystemSleepModeRequest (*gpc_cpu_slice_t slice*, *const gpc_cpu_mode_t mode*)

Parameters

<i>slice</i>	GPC CPU slice number.
<i>mode</i>	CPU mode. Refer to "gpc_cpu_mode_t".

49.4.11 static bool GPC_CM_GetSystemSleepModeStatus (*gpc_cpu_slice_t slice*, *uint32_t mask*) [inline], [static]

Parameters

<i>slice</i>	GPC CPU slice number.
<i>mask</i>	System sleep mode transition status mask, refer to "gpc_cm_system_sleep_mode_status_t".

Returns

Indicate the CPU's system sleep transition status.

49.4.12 void GPC_SS_EnableSystemSleepTransitionStep (GPC_SYS_SLEEP_CTRL_Type * *base*, gpc_ss_tran_step_t *step*, bool *enable*)

param base GPC system sleep controller base address. param step step type, refer to "gpc_ss_tran_step_t". param enable Used to control the transition step.

- **true** This step is enabled.
- **false** This step is disabled, GPC will skip this step and not send any request.

49.4.13 static void GPC_SS_SoftwareTriggerPMICStandby (GPC_SYS_SLEEP_CTRL_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	PMIC module base address.
<i>enable</i>	Trigger on/off PMIC standby. <ul style="list-style-type: none"> • true Trigger PMIC standby ON. • false Trigger PMIC standby OFF.

49.4.14 static void GPC_SS_SystemSleepTriggerPMICStandby (GPC_SYS_SLEEP_CTRL_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	PMIC module base address.
<i>enable</i>	Assert PMIC standby request or not. <ul style="list-style-type: none">• true Assert PMIC_STBY_REQ when system sleep is entered.• false Do not assert PMIC_STBY_REQ when system sleep is entered.

Chapter 50

Asrc_edma_driver

50.1 Overview

Data Structures

- struct `asrc_p2p_edma_config_t`
destination peripheral configuration [More...](#)
- struct `asrc_in_edma_handle_t`
@ brief asrc in edma handler [More...](#)
- struct `asrc_out_edma_handle_t`
@ brief asrc out edma handler [More...](#)
- struct `asrc_edma_handle_t`
ASRC DMA transfer handle. [More...](#)

Macros

- #define `ASRC_XFER_IN_QUEUE_SIZE` 4U
<

Typedefs

- typedef void(* `asrc_edma_callback_t`)(ASRC_Type *base, asrc_edma_handle_t *handle, `status_t` status, void *userData)
ASRC eDMA transfer callback function for finish and error.
- typedef void(* `asrc_start_peripheral_t`)(bool start)
ASRC trigger peripheral function pointer.

Driver version

- #define `FSL_ASRC_EDMA_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 0)`)
Version 2.2.0.

eDMA Transactional

- void `ASRC_TransferInCreateHandleEDMA` (ASRC_Type *base, asrc_edma_handle_t *handle, `asrc_channel_pair_t` channelPair, `asrc_edma_callback_t` callback, `edma_handle_t` *inDmaHandle, const `asrc_p2p_edma_config_t` *periphConfig, void *userData)
Initializes the ASRC IN eDMA handle.
- void `ASRC_TransferOutCreateHandleEDMA` (ASRC_Type *base, asrc_edma_handle_t *handle, `asrc_channel_pair_t` channelPair, `asrc_edma_callback_t` callback, `edma_handle_t` *outDmaHandle, const `asrc_p2p_edma_config_t` *periphConfig, void *userData)
Initializes the ASRC OUT eDMA handle.
- `status_t ASRC_TransferSetChannelPairConfigEDMA` (ASRC_Type *base, asrc_edma_handle_t *handle, `asrc_channel_config_t` *asrcConfig, uint32_t inSampleRate, uint32_t outSampleRate)

- `uint32_t ASRC_GetOutSamplesSizeEDMA` (`ASRC_Type *base, asrc_edma_handle_t *handle, uint32_t inSampleRate, uint32_t outSampleRate, uint32_t inSamplesize)`
Configures the ASRC P2P channel pair.
- `status_t ASRC_TransferEDMA` (`ASRC_Type *base, asrc_edma_handle_t *handle, asrc_transfer_t *xfer)`
Get output sample buffer size can be transferred by edma.
- `void ASRC_TransferInAbortEDMA` (`ASRC_Type *base, asrc_edma_handle_t *handle)`
Aborts a ASRC IN transfer using eDMA.
- `void ASRC_TransferOutAbortEDMA` (`ASRC_Type *base, asrc_edma_handle_t *handle)`
Aborts a ASRC OUT transfer using eDMA.
- `void ASRC_TransferInTerminalEDMA` (`ASRC_Type *base, asrc_edma_handle_t *handle)`
Terminate In ASRC Convert.
- `void ASRC_TransferOutTerminalEDMA` (`ASRC_Type *base, asrc_edma_handle_t *handle)`
Terminate Out ASRC Convert.

50.2 Data Structure Documentation

50.2.1 struct asrc_p2p_edma_config_t

Data Fields

- `asrc_start_peripheral_t startPeripheral`
trigger peripheral start

50.2.2 struct asrc_in_edma_handle_t

Data Fields

- `edma_handle_t * inDmaHandle`
DMA handler for ASRC in.
- `uint8_t tcd [ASRC_XFER_IN_QUEUE_SIZE+1U]*sizeof(edma_tcd_t)]`
TCD pool for eDMA send.
- `uint32_t sampleWidth`
input data width
- `uint32_t fifoThreshold`
ASRC input fifo threshold.
- `uint32_t * asrcQueue [ASRC_XFER_IN_QUEUE_SIZE]`
Transfer queue storing queued transfer.
- `size_t transferSize [ASRC_XFER_IN_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 state`
Internal state for ASRC eDMA transfer.
- `const asrc_p2p_edma_config_t * peripheralConfig`

peripheral configuration pointer

Field Documentation

- (1) `uint8_t asrc_in_edma_handle_t::tcd[(ASRC_XFER_IN_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`
- (2) `uint32_t* asrc_in_edma_handle_t::asrcQueue[ASRC_XFER_IN_QUEUE_SIZE]`
- (3) `volatile uint8_t asrc_in_edma_handle_t::queueUser`

50.2.3 struct asrc_out_edma_handle_t

Data Fields

- `edma_handle_t * outDmaHandle`
DMA handler for ASRC out.
- `uint8_t tcd [(ASRC_XFER_OUT_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`
TCD pool for eDMA send.
- `uint32_t sampleWidth`
output data width
- `uint32_t fifoThreshold`
ASRC output fifo threshold.
- `uint32_t * asrcQueue [ASRC_XFER_OUT_QUEUE_SIZE]`
Transfer queue storing queued transfer.
- `size_t transferSize [ASRC_XFER_OUT_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 state`
Internal state for ASRC eDMA transfer.
- `const asrc_p2p_edma_config_t * peripheralConfig`
peripheral configuration pointer

Field Documentation

- (1) `uint8_t asrc_out_edma_handle_t::tcd[(ASRC_XFER_OUT_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`
- (2) `uint32_t* asrc_out_edma_handle_t::asrcQueue[ASRC_XFER_OUT_QUEUE_SIZE]`
- (3) `volatile uint8_t asrc_out_edma_handle_t::queueUser`

50.2.4 struct _asrc_edma_handle**Data Fields**

- `asrc_in_edma_handle_t in`
asrc in handler
- `asrc_out_edma_handle_t out`
asrc out handler
- `asrc_channel_pair_t channelPair`
channel pair
- `void *userData`
User callback parameter.
- `asrc_edma_callback_t callback`
Callback for users while transfer finish or error occurs.

50.3 Macro Definition Documentation**50.3.1 #define ASRC_XFER_IN_QUEUE_SIZE 4U**

ASRC IN edma QUEUE size

50.4 Function Documentation
**50.4.1 void ASRC_TransferInCreateHandleEDMA (ASRC_Type * base,
 asrc_edma_handle_t * handle, asrc_channel_pair_t channelPair,
 asrc_edma_callback_t callback, edma_handle_t * inDmaHandle, const
 asrc_p2p_edma_config_t * periphConfig, void * userData)**

This function initializes the ASRC DMA handle, which can be used for other ASRC transactional APIs. Usually, for a specified ASRC channel pair, call this API once to get the initialized handle.

Parameters

<i>base</i>	ASRC base pointer.
<i>channelPair</i>	ASRC channel pair
<i>handle</i>	ASRC eDMA handle pointer.
<i>callback</i>	Pointer to user callback function.
<i>inDmaHandle</i>	DMA handler for ASRC in.
<i>periphConfig</i>	peripheral configuration.
<i>userData</i>	User parameter passed to the callback function.

**50.4.2 void ASRC_TransferOutCreateHandleEDMA (ASRC_Type * *base*,
asrc_edma_handle_t * *handle*, *asrc_channel_pair_t* *channelPair*,
asrc_edma_callback_t *callback*, *edma_handle_t* * *outDmaHandle*, const
asrc_p2p_edma_config_t * *periphConfig*, *void* * *userData*)**

This function initializes the ASRC DMA handle, which can be used for other ASRC transactional APIs. Usually, for a specified ASRC channel pair, call this API once to get the initialized handle.

Parameters

<i>base</i>	ASRC base pointer.
<i>channelPair</i>	ASRC channel pair
<i>handle</i>	ASRC eDMA handle pointer.
<i>callback</i>	Pointer to user callback function.
<i>outDmaHandle</i>	DMA handler for ASRC out.
<i>periphConfig</i>	peripheral configuration.
<i>userData</i>	User parameter passed to the callback function.

**50.4.3 status_t ASRC_TransferSetChannelPairConfigEDMA (ASRC_Type * *base*,
asrc_edma_handle_t * *handle*, *asrc_channel_pair_config_t* * *asrcConfig*,
uint32_t *inSampleRate*, *uint32_t* *outSampleRate*)**

Parameters

<i>base</i>	ASRC base pointer.
<i>handle</i>	ASRC eDMA handle pointer.
<i>asrcConfig</i>	asrc configurations.
<i>inSampleRate</i>	ASRC input sample rate.
<i>outSampleRate</i>	ASRC output sample rate.

50.4.4 uint32_t ASRC_GetOutSamplesSizeEDMA (ASRC_Type * *base*, asrc_edma_handle_t * *handle*, uint32_t *inSampleRate*, uint32_t *outSampleRate*, uint32_t *inSamplesize*)

Note

This API is depends on the ASRC output configuration, should be called after the ASRC_Transfer-SetChannelPairConfigEDMA.

Parameters

<i>base</i>	asrc base pointer.
<i>handle</i>	ASRC channel pair edma handle.
<i>inSampleRate</i>	input sample rate.
<i>outSampleRate</i>	output sample rate.
<i>inSamplesize</i>	input sampleS size.

Return values

<i>output</i>	buffer size in byte.
---------------	----------------------

50.4.5 status_t ASRC_TransferEDMA (ASRC_Type * *base*, asrc_edma_handle_t * *handle*, asrc_transfer_t * *xfer*)

Note

This interface returns immediately after the transfer initiates.

Parameters

<i>base</i>	ASRC base pointer.
<i>handle</i>	ASRC eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a ASRC eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_ASRCQueueFull</i>	ASRC EDMA driver queue is full.

50.4.6 void ASRC_TransferInAbortEDMA (ASRC_Type * *base*, asrc_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call ASRC_TransferTerminalP2PEDMA.

Parameters

<i>base</i>	ASRC base pointer.
<i>handle</i>	ASRC eDMA handle pointer.

50.4.7 void ASRC_TransferOutAbortEDMA (ASRC_Type * *base*, asrc_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call ASRC_TransferTerminalP2PEDMA.

Parameters

<i>base</i>	ASRC base pointer.
<i>handle</i>	ASRC eDMA handle pointer.

50.4.8 void ASRC_TransferInTerminalEDMA (**ASRC_Type** * *base*, **asrc_edma_handle_t** * *handle*)

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC_TransferAbortPP2PEDMA.

Parameters

<i>base</i>	ASRC base pointer.
<i>handle</i>	ASRC eDMA handle pointer.

50.4.9 void ASRC_TransferOutTerminalEDMA (ASRC_Type * *base*, asrc_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the asrc queue. If users only want to abort the current transfer slot, please call ASRC_TransferAbortPP2PEDMA.

Parameters

<i>base</i>	ASRC base pointer.
<i>handle</i>	ASRC eDMA handle pointer.

Chapter 51

I3c_master_edma_driver

51.1 Overview

Data Structures

- struct [i3c_master_edma_callback_t](#)
i3c master callback functions. [More...](#)
- struct [i3c_master_edma_handle_t](#)
Driver handle for master EDMA APIs. [More...](#)

Master DMA

- void [I3C_MasterTransferCreateHandleEDMA](#) (I3C_Type *base, i3c_master_edma_handle_t *handle, const i3c_master_edma_callback_t *callback, void *userData, edma_handle_t *rxDmaHandle, edma_handle_t *txDmaHandle)
Create a new handle for the I3C master DMA APIs.
- status_t [I3C_MasterTransferEDMA](#) (I3C_Type *base, i3c_master_edma_handle_t *handle, i3c_master_transfer_t *transfer)
Performs a non-blocking DMA-based transaction on the I3C bus.
- status_t [I3C_MasterTransferGetCountEDMA](#) (I3C_Type *base, i3c_master_edma_handle_t *handle, size_t *count)
Returns number of bytes transferred so far.
- void [I3C_MasterTransferAbortEDMA](#) (I3C_Type *base, i3c_master_edma_handle_t *handle)
Terminates a non-blocking I3C master transmission early.
- void [I3C_MasterTransferEDMAHandleIRQ](#) (I3C_Type *base, void *i3cHandle)
Reusable routine to handle master interrupts.

51.2 Data Structure Documentation

51.2.1 struct i3c_master_edma_callback_t

Data Fields

- void(* [slave2Master](#))(I3C_Type *base, void *userData)
Transfer complete callback.
- void(* [ibiCallback](#))(I3C_Type *base, i3c_master_edma_handle_t *handle, [i3c_ibi_type_t](#) ibiType, [i3c_ibi_state_t](#) ibiState)
IBI event callback.
- void(* [transferComplete](#))(I3C_Type *base, i3c_master_edma_handle_t *handle, [status_t](#) status, void *userData)
Transfer complete callback.

51.2.2 struct _i3c_master_edma_handle

Note

The contents of this structure are private and subject to change.

Data Fields

- I3C_Type * **base**
I3C base pointer.
- uint8_t **state**
Transfer state machine current state.
- uint32_t **transferCount**
Indicates progress of the transfer.
- uint8_t **subaddressBuffer** [4]
Saving subaddress command.
- uint8_t **subaddressCount**
Saving command count.
- i3c_master_transfer_t **transfer**
Copy of the current transfer info.
- i3c_master_edma_callback_t **callback**
Callback function pointer.
- void * **userData**
Application data passed to callback.
- edma_handle_t * **rxDmaHandle**
Handle for receive DMA channel.
- edma_handle_t * **txDmaHandle**
Handle for transmit DMA channel.
- uint8_t **ibiAddress**
Slave address which request IBI.
- uint8_t * **ibiBuff**
Pointer to IBI buffer to keep ibi bytes.
- size_t **ibiPayloadSize**
IBI payload size.
- i3c_ibi_type_t **ibiType**
IBI type.

Field Documentation

- (1) I3C_Type* i3c_master_edma_handle_t::base
- (2) uint8_t i3c_master_edma_handle_t::state
- (3) uint8_t i3c_master_edma_handle_t::subaddressBuffer[4]
- (4) uint8_t i3c_master_edma_handle_t::subaddressCount
- (5) i3c_master_transfer_t i3c_master_edma_handle_t::transfer
- (6) i3c_master_edma_callback_t i3c_master_edma_handle_t::callback
- (7) void* i3c_master_edma_handle_t::userData
- (8) edma_handle_t* i3c_master_edma_handle_t::rxDmaHandle
- (9) edma_handle_t* i3c_master_edma_handle_t::txDmaHandle
- (10) uint8_t i3c_master_edma_handle_t::ibiAddress
- (11) uint8_t* i3c_master_edma_handle_t::ibiBuff
- (12) size_t i3c_master_edma_handle_t::ibiPayloadSize
- (13) i3c_ibit_type_t i3c_master_edma_handle_t::ibiType

51.3 Function Documentation

**51.3.1 void I3C_MasterTransferCreateHandleEDMA (I3C_Type * *base*,
i3c_master_edma_handle_t * *handle*, const *i3c_master_edma_callback_t* *
callback, void * *userData*, *edma_handle_t* * *rxDmaHandle*, *edma_handle_t*
* *txDmaHandle*)**

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_MasterTransferAbortDMA() API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

<i>base</i>	The I3C peripheral base address.
<i>handle</i>	Pointer to the I3C master driver handle.
<i>callback</i>	User provided pointer to the asynchronous callback function.
<i>userData</i>	User provided pointer to the application callback data.
<i>rxDmaHandle</i>	Handle for the DMA receive channel. Created by the user prior to calling this function.
<i>txDmaHandle</i>	Handle for the DMA transmit channel. Created by the user prior to calling this function.

51.3.2 status_t I3C_MasterTransferEDMA (*I3C_Type* * *base*, *i3c_master_edma_handle_t* * *handle*, *i3c_master_transfer_t* * *transfer*)

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

<i>base</i>	The I3C peripheral base address.
<i>handle</i>	Pointer to the I3C master driver handle.
<i>transfer</i>	The pointer to the transfer descriptor.

Return values

<i>kStatus_Success</i>	The transaction was started successfully.
<i>kStatus_I3C_Busy</i>	Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

51.3.3 status_t I3C_MasterTransferGetCountEDMA (*I3C_Type* * *base*, *i3c_master_edma_handle_t* * *handle*, *size_t* * *count*)

Parameters

	<i>base</i>	The I3C peripheral base address.
--	-------------	----------------------------------

	<i>handle</i>	Pointer to the I3C master driver handle.
out	<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_Success</i>	
<i>kStatus_NoTransferIn-Progress</i>	There is not a DMA transaction currently in progress.

51.3.4 void I3C_MasterTransferAbortEDMA (*I3C_Type* * *base*, *i3c_master_edma_handle_t* * *handle*)

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

Parameters

<i>base</i>	The I3C peripheral base address.
<i>handle</i>	Pointer to the I3C master driver handle.

51.3.5 void I3C_MasterTransferEDMAHandleIRQ (*I3C_Type* * *base*, *void* * *i3cHandle*)

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

<i>base</i>	The I3C peripheral base address.
<i>i3cHandle</i>	Pointer to the I3C master DMA driver handle.

Chapter 52

i3c_slave_edma_driver

52.1 Overview

Data Structures

- struct [i3c_slave_edma_transfer_t](#)
I3C slave transfer structure. [More...](#)
- struct [i3c_slave_edma_handle_t](#)
I3C slave edma handle structure. [More...](#)

Typedefs

- typedef void(* [i3c_slave_edma_callback_t](#))(I3C_Type *base, i3c_slave_edma_transfer_t *transfer, void *userData)
Slave event callback function pointer type.

Slave DMA

- void [I3C_SlaveTransferCreateHandleEDMA](#) (I3C_Type *base, i3c_slave_edma_handle_t *handle, i3c_slave_edma_callback_t callback, void *userData, [edma_handle_t](#) *rxDmaHandle, [edma_handle_t](#) *txDmaHandle)
Create a new handle for the I3C slave DMA APIs.
- status_t [I3C_SlaveTransferEDMA](#) (I3C_Type *base, i3c_slave_edma_handle_t *handle, [i3c_slave_edma_transfer_t](#) *transfer, uint32_t eventMask)
Prepares for a non-blocking DMA-based transaction on the I3C bus.
- void [I3C_SlaveTransferAbortEDMA](#) (I3C_Type *base, i3c_slave_edma_handle_t *handle)
Abort a slave edma non-blocking transfer in a early time.
- void [I3C_SlaveTransferEDMAHandleIRQ](#) (I3C_Type *base, void *i3cHandle)
Reusable routine to handle slave interrupts.

52.2 Data Structure Documentation

52.2.1 struct i3c_slave_edma_transfer_t

Data Fields

- uint32_t **event**
Reason the callback is being invoked.
- uint8_t * **txData**
Transfer buffer.
- size_t **txDataSize**
Transfer size.
- uint8_t * **rxData**

- *Transfer buffer.*
`size_t rxDataSize`
- *Transfer size.*
`status_t completionStatus`
- *Success or error code describing how the transfer completed.*

Field Documentation

- (1) `uint32_t i3c_slave_edma_transfer_t::event`
- (2) `status_t i3c_slave_edma_transfer_t::completionStatus`

Only applies for [kI3C_SlaveCompletionEvent](#).

52.2.2 struct _i3c_slave_edma_handle

Note

The contents of this structure are private and subject to change.

Data Fields

- `I3C_Type * base`
I3C base pointer.
- `i3c_slave_edma_transfer_t transfer`
I3C slave transfer copy.
- `bool isBusy`
Whether transfer is busy.
- `bool wasTransmit`
Whether the last transfer was a transmit.
- `uint32_t eventMask`
Mask of enabled events.
- `i3c_slave_edma_callback_t callback`
Callback function called at transfer event.
- `edma_handle_t * rxDmaHandle`
Handle for receive DMA channel.
- `edma_handle_t * txDmaHandle`
Handle for transmit DMA channel.
- `void * userData`
Callback parameter passed to callback.

Field Documentation

- (1) `I3C_Type* i3c_slave_edma_handle_t::base`
- (2) `i3c_slave_edma_transfer_t i3c_slave_edma_handle_t::transfer`
- (3) `bool i3c_slave_edma_handle_t::isBusy`
- (4) `bool i3c_slave_edma_handle_t::wasTransmit`
- (5) `uint32_t i3c_slave_edma_handle_t::eventMask`
- (6) `i3c_slave_edma_callback_t i3c_slave_edma_handle_t::callback`
- (7) `edma_handle_t* i3c_slave_edma_handle_t::rxDmaHandle`
- (8) `edma_handle_t* i3c_slave_edma_handle_t::txDmaHandle`
- (9) `void* i3c_slave_edma_handle_t::userData`

52.3 Typedef Documentation

52.3.1 `typedef void(* i3c_slave_edma_callback_t)(I3C_Type *base, i3c_slave_edma_transfer_t *transfer, void *userData)`

This callback is used only for the slave DMA transfer API.

Parameters

<i>base</i>	Base address for the I3C instance on which the event occurred.
<i>handle</i>	Pointer to slave DMA transfer handle.
<i>transfer</i>	Pointer to transfer descriptor containing values passed to and/or from the callback.
<i>userData</i>	Arbitrary pointer-sized value passed from the application.

52.4 Function Documentation

52.4.1 `void I3C_SlaveTransferCreateHandleEDMA (I3C_Type * base, i3c_slave_edma_handle_t * handle, i3c_slave_edma_callback_t callback, void * userData, edma_handle_t * rxDmaHandle, edma_handle_t * txDmaHandle)`

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the `I3C_SlaveTransferAbortDMA()` API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the `txDmaHandle` parameter is ignored and may be set to NULL.

Parameters

<i>base</i>	The I3C peripheral base address.
<i>handle</i>	Pointer to the I3C slave driver handle.
<i>callback</i>	User provided pointer to the asynchronous callback function.
<i>userData</i>	User provided pointer to the application callback data.
<i>rxDmaHandle</i>	Handle for the DMA receive channel. Created by the user prior to calling this function.
<i>txDmaHandle</i>	Handle for the DMA transmit channel. Created by the user prior to calling this function.

52.4.2 status_t I3C_SlaveTransferEDMA (*I3C_Type* * *base*, *i3c_slave_edma_handle_t* * *handle*, *i3c_slave_edma_transfer_t* * *transfer*, *uint32_t eventMask*)

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

<i>base</i>	The I3C peripheral base address.
<i>handle</i>	Pointer to the I3C slave driver handle.
<i>transfer</i>	The pointer to the transfer descriptor.
<i>eventMask</i>	Bit mask formed by OR'ing together <i>i3c_slave_transfer_event_t</i> enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

Return values

<i>kStatus_Success</i>	The transaction was started successfully.
<i>kStatus_I3C_Busy</i>	Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

<i>kStatus_Fail</i>	The transaction can't be set.
---------------------	-------------------------------

52.4.3 void I3C_SlaveTransferAbortEDMA (*I3C_Type* * *base*, *i3c_slave_edma_handle_t* * *handle*)

Parameters

<i>base</i>	I3C peripheral base address
<i>handle</i>	pointer to <i>i3c_slave_edma_handle_t</i> structure

52.4.4 void I3C_SlaveTransferEDMAHandleIRQ (*I3C_Type* * *base*, *void* * *i3cHandle*)

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

<i>base</i>	The I3C peripheral base address.
<i>i3cHandle</i>	Pointer to the I3C slave DMA driver handle.

Chapter 53

Xbar

53.1 Overview

Data Structures

- struct `xbar_info_t`
Find the instance index from base address and register offset mappings. [More...](#)
- struct `xbar_control_config_t`
Defines the configuration structure of the XBAR control register. [More...](#)

Enumerations

- enum `xbar_active_edge_t` {
 `kXBAR_EdgeNone` = 0U,
 `kXBAR_EdgeRising` = 1U,
 `kXBAR_EdgeFalling` = 2U,
 `kXBAR_EdgeRisingAndFalling` = 3U }
XBAR active edge for detection.
- enum `xbar_request_t` {
 `kXBAR_RequestDisable` = 0U,
 `kXBAR_RequestDMAEnable` = 1U,
 `kXBAR_RequestInterruptEnable` = 2U }
Defines the XBAR DMA and interrupt configurations.

XBAR functional Operation

- void `XBAR_Init` (`xbar_instance_t` xbarInstance)
Initializes the XBAR modules.
- void `XBAR_Deinit` (`xbar_instance_t` xbarInstance)
Shutdown the XBAR modules.
- `status_t XBAR_SetSignalsConnection` (`xbar_input_signal_t` input, `xbar_output_signal_t` output)
Set connection between the selected XBAR_IN[] input and the XBAR_OUT[*] output signal.*
- `status_t XBAR_ClearOutputStatusFlag` (`xbar_output_signal_t` output)
Clears the edge detection status flags.
- `status_t XBAR_GetOutputStatusFlag` (`xbar_output_signal_t` output, `bool` *flag)
Gets the active edge detection status.
- `status_t XBAR_SetOutputSignalConfig` (`xbar_output_signal_t` output, const `xbar_control_config_t` *controlConfig)
Configures the XBAR control register.

53.2 Data Structure Documentation

53.2.1 struct xbar_info_t

53.2.2 struct xbar_control_config_t

This structure keeps the configuration of XBAR control register for one output. Control registers are available only for a few outputs. Not every XBAR module has control registers.

Data Fields

- `xbar_active_edge_t activeEdge`
Active edge to be detected.
- `xbar_request_t requestType`
Selects DMA/Interrupt request.

Field Documentation

(1) `xbar_active_edge_t xbar_control_config_t::activeEdge`

(2) `xbar_request_t xbar_control_config_t::requestType`

53.3 Enumeration Type Documentation

53.3.1 enum xbar_active_edge_t

Enumerator

`kXBAR_EdgeNone` Edge detection status bit never asserts.

`kXBAR_EdgeRising` Edge detection status bit asserts on rising edges.

`kXBAR_EdgeFalling` Edge detection status bit asserts on falling edges.

`kXBAR_EdgeRisingAndFalling` Edge detection status bit asserts on rising and falling edges.

53.3.2 enum xbar_request_t

Enumerator

`kXBAR_RequestDisable` Interrupt and DMA are disabled.

`kXBAR_RequestDMAEnable` DMA enabled, interrupt disabled.

`kXBAR_RequestInterruptEnable` Interrupt enabled, DMA disabled.

53.4 Function Documentation

53.4.1 void XBAR_Init (`xbar_instance_t xbarInstance`)

This function un-gates the XBAR clock.

Parameters

<i>xbarInstance</i>	XBAR peripheral address.
---------------------	--------------------------

53.4.2 void XBAR_Deinit(*xbar_instance_t xbarInstance*)

This function disables XBAR clock.

Parameters

<i>xbarInstance</i>	XBAR peripheral address.
---------------------	--------------------------

53.4.3 status_t XBAR_SetSignalsConnection(*xbar_input_signal_t input*, *xbar_output_signal_t output*)

This function connects the XBAR input to the selected XBAR output. If more than one XBAR module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBAR_SetSignalsConnection(kXBAR_DSC1_InputLogicLow,
                           kXBAR_DSC1_OutputTriggerSyncIn0);
```

Parameters

<i>input</i>	XBAR input signal.
<i>output</i>	XBAR output signal.

Return values

<i>kStatus_Success</i>	Signal connection set successfully.
<i>kStatus_InvalidArgument</i>	Failed because of invalid argument.

53.4.4 status_t XBAR_ClearOutputStatusFlag(*xbar_output_signal_t output*)

Parameters

<i>output</i>	XBAR output signal.
---------------	---------------------

Return values

<i>kStatus_Success</i>	Signal connection set successfully.
<i>kStatus_InvalidArgument</i>	Failed because of invalid argument.

53.4.5 status_t XBAR_GetOutputStatusFlag (*xbar_output_signal_t output, bool * flag*)

This function gets the active edge detect status of all XBAR_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBAR_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

<i>output</i>	XBAR output signal.
<i>flag</i>	get XBAR output status flag.

Return values

<i>kStatus_Success</i>	Signal connection set successfully.
<i>kStatus_InvalidArgument</i>	Failed because of invalid argument.

53.4.6 status_t XBAR_SetOutputSignalConfig (*xbar_output_signal_t output, const xbar_control_config_t * controlConfig*)

This function configures an XBAR control register. The active edge detection and the DMA/IRQ function on the corresponding XBAR output can be set.

Example:

```
xbar_control_config_t userConfig;
userConfig.activeEdge = kXBAR_EdgeRising;
userConfig.requestType = kXBAR_RequestInterruptEnable;
XBAR_SetOutputSignalConfig(kXBARA_OutputDMAMUX18, &userConfig);
```

Parameters

<i>output</i>	XBAR output signal.
<i>controlConfig</i>	Pointer to structure that keeps configuration of control register.

Return values

<i>kStatus_Success</i>	Signal connection set successfully.
<i>kStatus_InvalidArgument</i>	Failed because of invalid argument.

53.5 Wm8962

53.5.1 Overview

Data Structures

- struct `wm8962_audio_format_t`
wm8962 audio format More...
- struct `wm8962_fll_clk_config_t`
wm8962 master system clock configuration More...
- struct `wm8962_route_config_t`
WM8962 data route configurations. More...
- struct `wm8962_config_t`
Initialize structure of WM8962. More...
- struct `wm8962_handle_t`
wm8962 codec handler More...

Macros

- #define `WM8962_I2C_HANDLER_SIZE` CODEC_I2C_MASTER_HANDLER_SIZE
wm8962 handle size
- #define `WM8962_LINVOL` 0x0U
Define the register address of WM8962.
- #define `WM8962_CACHEREGNUM` 56U
Cache register number.
- #define `WM8962_CLOCK2_BCLK_DIV_MASK` 0xFU
WM8962 CLOCK2 bits.
- #define `WM8962_IFACE0_FORMAT_MASK` 0x13U
WM8962_IFACE0 FORMAT bits.
- #define `WM8962_IFACE0_WL_MASK` 0x0CU
WM8962_IFACE0 WL bits.
- #define `WM8962_IFACE1_LRP_MASK` 0x10U
WM8962_IFACE1 LRP bit.
- #define `WM8962_IFACE1_DLRSWAP_MASK` 0x20U
WM8962_IFACE1 DLRSWAP bit.
- #define `WM8962_IFACE1_MS_MASK` 0x40U
WM8962_IFACE1 MS bit.
- #define `WM8962_IFACE1_BCLKINV_MASK` 0x80U
WM8962_IFACE1 BCLKINV bit.
- #define `WM8962_IFACE1_ALRSWAP_MASK` 0x100U
WM8962_IFACE1 ALRSWAP bit.
- #define `WM8962_POWER1_VREF_MASK` 0x40U
WM8962_POWER1.
- #define `WM8962_POWER2_DACL_MASK` 0x100U
WM8962_POWER2.
- #define `WM8962_I2C_ADDR` (0x34 >> 1U)
WM8962 I2C address.
- #define `WM8962_I2C_BAUDRATE` (100000U)
WM8962 I2C baudrate.
- #define `WM8962_ADC_MAX_VOLUME_vVALUE` 0xFFU

WM8962 maximum volume value.

Enumerations

- enum {

kWM8962_InputMixerSourceInput2 = 4U,

kWM8962_InputMixerSourceInput3 = 2U,

kWM8962_InputMixerSourceInputPGA = 1U }

wm8962 input mixer source.
- enum {

kWM8962_OutputMixerDisabled = 0U,

kWM8962_OutputMixerSourceInput4Right = 1U,

kWM8962_OutputMixerSourceInput4Left = 2U,

kWM8962_OutputMixerSourceRightInputMixer = 4U,

kWM8962_OutputMixerSourceLeftInputMixer = 8U,

kWM8962_OutputMixerSourceRightDAC = 0x10U,

kWM8962_OutputMixerSourceLeftDAC = 0x20U }

wm8962 output mixer source.
- enum **wm8962_module_t** {

kWM8962_ModuleADC = 0,

kWM8962_ModuleDAC = 1,

kWM8962_ModuleMICB = 4,

kWM8962_ModuleMIC = 5,

kWM8962_ModuleLineIn = 6,

kWM8962_ModuleHeadphone = 7,

kWM8962_ModuleSpeaker = 8,

kWM8962_ModuleHeadphoneMixer = 9,

kWM8962_ModuleSpeakerMixer = 10 }

Modules in WM8962 board.
- enum {

kWM8962_HeadphoneLeft = 1,

kWM8962_HeadphoneRight = 2,

kWM8962_SpeakerLeft = 4,

kWM8962_SpeakerRight = 8 }

wm8962 play channel
- enum **wm8962_protocol_t** {

kWM8962_BusPCMA = 4,

kWM8962_BusPCMB = 3,

kWM8962_BusI2S = 2,

kWM8962_BusLeftJustified = 1,

kWM8962_BusRightJustified = 0 }

The audio data transfer protocol choice.
- enum **wm8962_input_pga_source_t** {

- ```
kWM8962_InputPGAInput1 = 8,
kWM8962_InputPGAInput2 = 4,
kWM8962_InputPGAInput3 = 2,
kWM8962_InputPGAInput4 = 1 }
```

*wm8962 input source*
- enum `wm8962_output_pga_source_t` {
 

```
kWM8962_OutputPGASourceMixer = 0,
kWM8962_OutputPGASourceDAC = 1 }
```

*wm8962 input source*
- enum {
 

```
kWM8962_AudioSampleRate8KHz = 8000U,
kWM8962_AudioSampleRate11025Hz = 11025U,
kWM8962_AudioSampleRate12KHz = 12000U,
kWM8962_AudioSampleRate16KHz = 16000U,
kWM8962_AudioSampleRate22050Hz = 22050U,
kWM8962_AudioSampleRate24KHz = 24000U,
kWM8962_AudioSampleRate32KHz = 32000U,
kWM8962_AudioSampleRate44100Hz = 44100U,
kWM8962_AudioSampleRate48KHz = 48000U,
kWM8962_AudioSampleRate88200Hz = 88200U,
kWM8962_AudioSampleRate96KHz = 96000U }
```

*audio sample rate definition*
- enum {
 

```
kWM8962_AudioBitWidth16bit = 16U,
kWM8962_AudioBitWidth20bit = 20U,
kWM8962_AudioBitWidth24bit = 24U,
kWM8962_AudioBitWidth32bit = 32U }
```

*audio bit width*
- enum `wm8962_fllclk_source_t` {
 

```
kWM8962_FLLClkSourceMCLK = 0U,
kWM8962_FLLClkSourceBCLK = 1U }
```

*wm8962 fll clock source*
- enum `wm8962_sysclk_source_t` {
 

```
kWM8962_SysClkSourceMclk = 0U,
kWM8962_SysClkSourceFLL = 1U }
```

*wm8962 sysclk source*

## Functions

- `status_t WM8962_Init (wm8962_handle_t *handle, const wm8962_config_t *config)`  
*WM8962 initialize function.*
- `status_t WM8962_Deinit (wm8962_handle_t *handle)`  
*Deinit the WM8962 codec.*
- `status_t WM8962_SetDataRoute (wm8962_handle_t *handle, const wm8962_route_config_t *route)`  
*Set audio data route in WM8962.*

- `status_t WM8962_SetProtocol (wm8962_handle_t *handle, wm8962_protocol_t protocol)`  
*Set the audio transfer protocol.*
- `status_t WM8962_SetModuleVolume (wm8962_handle_t *handle, wm8962_module_t module, uint32_t volume)`  
*Set the volume of different modules in WM8962.*
- `uint32_t WM8962_GetModuleVolume (wm8962_handle_t *handle, wm8962_module_t module)`  
*Get the volume of different modules in WM8962.*
- `status_t WM8962_SetModuleMute (wm8962_handle_t *handle, wm8962_module_t module, bool isEnabled)`  
*Mute modules in WM8962.*
- `status_t WM8962_SetModulePower (wm8962_handle_t *handle, wm8962_module_t module, bool isEnabled)`  
*Enable/disable expected devices.*
- `status_t WM8962_ConfigDataFormat (wm8962_handle_t *handle, uint32_t sysclk, uint32_t sample_rate, uint32_t bits)`  
*Configure the data format of audio data.*
- `status_t WM8962_WriteReg (wm8962_handle_t *handle, uint16_t reg, uint16_t val)`  
*Write register to WM8962 using I2C.*
- `status_t WM8962_ReadReg (wm8962_handle_t *handle, uint16_t reg, uint16_t *val)`  
*Read register from WM8962 using I2C.*
- `status_t WM8962_ModifyReg (wm8962_handle_t *handle, uint16_t reg, uint16_t mask, uint16_t val)`  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_WM8962_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`  
*CLOCK driver version 2.1.3.*

### 53.5.2 Data Structure Documentation

#### 53.5.2.1 struct `wm8962_audio_format_t`

##### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*

### 53.5.2.2 struct `wm8962_fll_clk_config_t`

#### Data Fields

- `wm8962_fllclk_source_t fllClockSource`  
*fll clock source*
- `uint32_t fllReferenceClockFreq`  
*external input frequency*
- `uint32_t fllOutputFreq`  
*FLL output frequency value.*

### 53.5.2.3 struct `wm8962_route_config_t`

#### Data Fields

- `bool enableLoopBack`  
*enable loopback: ADC->DAC directly*
- `wm8962_input_pga_source_t leftInputPGASource`  
*Left input source for WM8962.*
- `uint32_t leftInputMixerSource`  
*left input MIXER source, combination value of `wm8962_input_mixer_source_t`*
- `wm8962_input_pga_source_t rightInputPGASource`  
*right input PGA source*
- `uint32_t rightInputMixerSource`  
*right input MIXER source, combination value of `wm8962_input_mixer_source_t`*
- `uint32_t leftSpeakerMixerSource`  
*speaker left MIXER source, combination value of `wm8962_output_mixer_source_t`*
- `wm8962_output_pga_source_t leftSpeakerPGASource`  
*speaker left PGA source*
- `uint32_t rightSpeakerMixerSource`  
*speaker right MIXER source, combination value of `wm8962_output_mixer_source_t`*
- `wm8962_output_pga_source_t rightSpeakerPGASource`  
*speaker left PGA source*
- `uint32_t leftHeadphoneMixerSource`  
*headphone left MIXER source, combination value of `wm8962_output_mixer_source_t`*
- `wm8962_output_pga_source_t leftHeadphonePGASource`  
*speaker left PGA source*
- `uint32_t rightHeadphoneMixerSource`  
*headphone left MIXER source, combination value of `wm8962_output_mixer_source_t`*
- `wm8962_output_pga_source_t rightHeadphonePGASource`  
*speaker left PGA source*

### 53.5.2.4 struct `wm8962_config_t`

#### Data Fields

- `wm8962_route_config_t route`  
*Audio data route.*
- `wm8962_protocol_t bus`

- *Audio transfer protocol.*
- `wm8962_audio_format_t` *format*  
    *Audio format.*
- `bool masterSlave`  
    *Master or slave.*
- `wm8962_sysclk_source_t sysclkSource`  
    *sysclk source*
- `wm8962_fll_clk_config_t fllClock`  
    *FLL clock configurations, shall be configured when masterSlave is true.*
- `uint8_t slaveAddress`  
    *wm8962 device address*
- `codec_i2c_config_t i2cConfig`  
    *i2c configuration*

## Field Documentation

(1) `wm8962_route_config_t` `wm8962_config_t::route`

(2) `bool` `wm8962_config_t::masterSlave`

true: master mode, false: slave mode

## 53.5.2.5 struct `wm8962_handle_t`

### Data Fields

- `const wm8962_config_t * config`  
    *wm8904 config pointer*
- `uint16_t volume` [`DEBUG_WM8962_MODULE_NUM`]  
    *Audio volume value.*
- `uint8_t i2cHandle` [`WM8962_I2C_HANDLER_SIZE`]  
    *i2c handle*

## 53.5.3 Macro Definition Documentation

53.5.3.1 `#define WM8962_LINVOL 0x0U`

53.5.3.2 `#define WM8962_I2C_ADDR (0x34 >> 1U)`

## 53.5.4 Enumeration Type Documentation

### 53.5.4.1 anonymous enum

Enumerator

`kWM8962_InputMixerSourceInput2` input mixer source input 2

`kWM8962_InputMixerSourceInput3` input mixer source input 3

*kWM8962\_InputMixerSourceInputPGA* input mixer source input PGA

#### 53.5.4.2 anonymous enum

Enumerator

*kWM8962\_OutputMixerDisabled* output mixer disabled  
*kWM8962\_OutputMixerSourceInput4Right* output mixer source input 4 left  
*kWM8962\_OutputMixerSourceInput4Left* output mixer source input 4 right  
*kWM8962\_OutputMixerSourceRightInputMixer* output mixer source left input mixer  
*kWM8962\_OutputMixerSourceLeftInputMixer* output mixer source right input mixer  
*kWM8962\_OutputMixerSourceRightDAC* output mixer source left DAC  
*kWM8962\_OutputMixerSourceLeftDAC* output mixer source Right DAC

#### 53.5.4.3 enum `wm8962_module_t`

Enumerator

*kWM8962\_ModuleADC* ADC module in WM8962.  
*kWM8962\_ModuleDAC* DAC module in WM8962.  
*kWM8962\_ModuleMICB* Mic bias.  
*kWM8962\_ModuleMIC* Input Mic.  
*kWM8962\_ModuleLineIn* Analog in PGA.  
*kWM8962\_ModuleHeadphone* Line out module.  
*kWM8962\_ModuleSpeaker* Speaker module.  
*kWM8962\_ModuleHeaphoneMixer* Output mixer.  
*kWM8962\_ModuleSpeakerMixer* Output mixer.

#### 53.5.4.4 anonymous enum

Enumerator

*kWM8962\_HeadphoneLeft* wm8962 headphone left channel  
*kWM8962\_HeadphoneRight* wm8962 headphone right channel  
*kWM8962\_SpeakerLeft* wm8962 speaker left channel  
*kWM8962\_SpeakerRight* wm8962 speaker right channel

#### 53.5.4.5 enum `wm8962_protocol_t`

WM8962 only supports I2S format and PCM format.

Enumerator

*kWM8962\_BusPCMA* PCMA mode.

*kWM8962\_BusPCMB* PCMB mode.

*kWM8962\_BusI2S* I2S type.

*kWM8962\_BusLeftJustified* Left justified mode.

*kWM8962\_BusRightJustified* Right justified mode.

### 53.5.4.6 enum `wm8962_input_pga_source_t`

Enumerator

*kWM8962\_InputPGASourceInput1* Input PGA source input1.

*kWM8962\_InputPGASourceInput2* Input PGA source input2.

*kWM8962\_InputPGASourceInput3* Input PGA source input3.

*kWM8962\_InputPGASourceInput4* Input PGA source input4.

### 53.5.4.7 enum `wm8962_output_pga_source_t`

Enumerator

*kWM8962\_OutputPGASourceMixer* Output PGA source mixer.

*kWM8962\_OutputPGASourceDAC* Output PGA source DAC.

### 53.5.4.8 anonymous enum

Enumerator

*kWM8962\_AudioSampleRate8KHz* Sample rate 8000 Hz.

*kWM8962\_AudioSampleRate11025Hz* Sample rate 11025 Hz.

*kWM8962\_AudioSampleRate12KHz* Sample rate 12000 Hz.

*kWM8962\_AudioSampleRate16KHz* Sample rate 16000 Hz.

*kWM8962\_AudioSampleRate22050Hz* Sample rate 22050 Hz.

*kWM8962\_AudioSampleRate24KHz* Sample rate 24000 Hz.

*kWM8962\_AudioSampleRate32KHz* Sample rate 32000 Hz.

*kWM8962\_AudioSampleRate44100Hz* Sample rate 44100 Hz.

*kWM8962\_AudioSampleRate48KHz* Sample rate 48000 Hz.

*kWM8962\_AudioSampleRate88200Hz* Sample rate 88200 Hz.

*kWM8962\_AudioSampleRate96KHz* Sample rate 96000 Hz.

### 53.5.4.9 anonymous enum

Enumerator

*kWM8962\_AudioBitWidth16bit* audio bit width 16

*kWM8962\_AudioBitWidth20bit* audio bit width 20

*kWM8962\_AudioBitWidth24bit* audio bit width 24

*kWM8962\_AudioBitWidth32bit* audio bit width 32

### 53.5.4.10 enum **wm8962\_fllclk\_source\_t**

Enumerator

*kWM8962\_FLLClkSourceMCLK* FLL clock source from MCLK.

*kWM8962\_FLLClkSourceBCLK* FLL clock source from BCLK.

### 53.5.4.11 enum **wm8962\_sysclk\_source\_t**

Enumerator

*kWM8962\_SysClkSourceMclk* sysclk source from external MCLK

*kWM8962\_SysClkSourceFLL* sysclk source from internal FLL

## 53.5 Function Documentation

### 53.5.5.1 status\_t **WM8962\_Init** ( **wm8962\_handle\_t \* handle**, **const wm8962\_config\_t \* config** )

The second parameter is NULL to WM8962 in this version. If users want to change the settings, they have to use `wm8962_write_reg()` or `wm8962_modify_reg()` to set the register value of WM8962. Note: If the `codec_config` is NULL, it would initialize WM8962 using default settings. The default setting: `codec_config->route = kWM8962_RoutePlaybackandRecord` `codec_config->bus = kWM8962_BusI2S` `codec_config->master = slave`

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>handle</i> | WM8962 handle structure.        |
| <i>config</i> | WM8962 configuration structure. |

### 53.5.5.2 status\_t **WM8962\_Deinit** ( **wm8962\_handle\_t \* handle** )

This function close all modules in WM8962 to save power.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>handle</i> | WM8962 handle structure pointer. |
|---------------|----------------------------------|

### 53.5.5.3 status\_t WM8962\_SetDataRoute ( *wm8962\_handle\_t \* handle, const wm8962\_route\_config\_t \* route* )

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules. Note: If a new route is set, the previous route would not work.

Parameters

|               |                             |
|---------------|-----------------------------|
| <i>handle</i> | WM8962 handle structure.    |
| <i>route</i>  | Audio data route in WM8962. |

### 53.5.5.4 status\_t WM8962\_SetProtocol ( *wm8962\_handle\_t \* handle, wm8962\_protocol\_t protocol* )

WM8960 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>handle</i>   | WM8960 handle structure.      |
| <i>protocol</i> | Audio data transfer protocol. |

### 53.5.5.5 status\_t WM8962\_SetModuleVolume ( *wm8962\_handle\_t \* handle, wm8962\_module\_t module, uint32\_t volume* )

This function would set the volume of WM8962 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Module:kWM8962\_ModuleADC, volume range value: 0 is mute, 1-255 is -97db to 30db Module:kWM8962\_ModuleDAC, volume range value: 0 is mute, 1-255 is -127db to 0db Module:kWM8962\_ModuleHP, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db Module:kWM8962\_ModuleLineIn, volume range value: 0 - 0x3F is -17.25db to 30db Module:kWM8962\_ModuleSpeaker, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>handle</i> | WM8962 handle structure.                                       |
| <i>module</i> | Module to set volume, it can be ADC, DAC, Headphone and so on. |
| <i>volume</i> | Volume value need to be set.                                   |

### 53.5.5.6 `uint32_t WM8962_GetModuleVolume ( wm8962_handle_t * handle, wm8962_module_t module )`

This function gets the volume of WM8962 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>handle</i> | WM8962 handle structure.                                       |
| <i>module</i> | Module to set volume, it can be ADC, DAC, Headphone and so on. |

Returns

Volume value of the module.

### 53.5.5.7 `status_t WM8962_SetModuleMute ( wm8962_handle_t * handle, wm8962_module_t module, bool isEnabled )`

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>handle</i>    | WM8962 handle structure.          |
| <i>module</i>    | Modules need to be mute.          |
| <i>isEnabled</i> | Mute or unmute, 1 represent mute. |

### 53.5.5.8 `status_t WM8962_SetModulePower ( wm8962_handle_t * handle, wm8962_module_t module, bool isEnabled )`

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8962 handle structure. |
|---------------|--------------------------|

|                  |                            |
|------------------|----------------------------|
| <i>module</i>    | Module expected to enable. |
| <i>isEnabled</i> | Enable or disable moudles. |

### 53.5.5.9 status\_t WM8962\_ConfigDataFormat ( *wm8962\_handle\_t \* handle, uint32\_t sysclk, uint32\_t sample\_rate, uint32\_t bits* )

This function would configure the registers about the sample rate, bit depths.

Parameters

|                    |                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>      | WM8962 handle structure pointer.                                                                                                          |
| <i>sysclk</i>      | system clock of the codec which can be generated by MCLK or PLL output.                                                                   |
| <i>sample_rate</i> | Sample rate of audio file running in WM8962. WM8962 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate. |
| <i>bits</i>        | Bit depth of audio file (WM8962 only supports 16bit, 20bit, 24bit and 32 bit in HW).                                                      |

### 53.5.5.10 status\_t WM8962\_WriteReg ( *wm8962\_handle\_t \* handle, uint16\_t reg, uint16\_t val* )

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>handle</i> | WM8962 handle structure.                |
| <i>reg</i>    | The register address in WM8962.         |
| <i>val</i>    | Value needs to write into the register. |

### 53.5.5.11 status\_t WM8962\_ReadReg ( *wm8962\_handle\_t \* handle, uint16\_t reg, uint16\_t \* val* )

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>handle</i> | WM8962 handle structure.        |
| <i>reg</i>    | The register address in WM8962. |

|            |                   |
|------------|-------------------|
| <i>val</i> | Value written to. |
|------------|-------------------|

### 53.5.5.12 **status\_t WM8962\_ModifyReg ( *wm8962\_handle\_t \* handle, uint16\_t reg, uint16\_t mask, uint16\_t val* )**

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>handle</i> | WM8962 handle structure.                                                         |
| <i>reg</i>    | The register address in WM8962.                                                  |
| <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.                                          |

**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.

